A Brief Survey of Self-Dual Codes

by

Rini Oktavia, S. Si., M. Si.

Report

Presented to the Faculty of the Graduate School
of the University of Texas at Austin
in Partial Fulfillment
of the Requirements
for the Degree of

Master of Arts

The University of Texas at Austin
August, 2009

The Report committee for Rini Oktavia
Certifies that this is the approved version of the following report:

A Brief Survey of Self-Dual Codes

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor: Jose F. Voloch

Geir T. Helleloid

A Brief Survey of Self-Dual Codes

by

Rini Oktavia, M.A.
The University of Texas at Austin, 2009
SUPERVISOR: Jose F. Voloch, Geir T. Helleloid

Abstract

This report is a survey of self-dual binary codes. We present the fundamental MacWilliams identity and Gleason's theorem on self-dual binary codes. We also examine the upper bound of minimum weights of self-dual binary codes using the extremal weight enumerator formula. We describe the shadow code of a self-dual code and the restrictions of the weight enumerator of the shadow code. Then using the restrictions, we calculate the weight enumerators of self-dual codes of length 38 and 40 and we obtain the known weight enumerators of this lengths. Finally, we investigate the Gaborit-Otmani experimental construction of self-dual binary codes. This construction involves a fixed orthogonal matrix, and we compare the result to the results obtained using other orthogonal matrices.

Acknowledgments

Foremost I would like to express my sincere gratitude to Dr. Geir Helleloid under whose supervision I choose this topic and began the survey. This report would not have been possible without Dr. Helleloid's continuous support and encouragement. His guidance helped me in all parts of the survey and writing the report. His critical comments directed me to be more insightful in citing the bibliography. His patience in correcting my earlier draft which had a lot of mistakes was very impressive. I could not have imagined having a better advisor for my report.

I also would like to express my gratitude to Dr. Jose F. Voloch, whose thoughtful advice often served to give me a sense of direction during my master studies. And I am grateful to the Ford Foundation International Fellowship Program for the support that they gave me in order to study in the US.

My sincere thanks also goes to Dr. John Tate, Dr. Jeffrey Vaaler, Dr. James W. Vick, and Dr. Hizir Sofyan for their trust and support for my professional development. I also would like to thank Dr. Lorenzo Sadun, Nancy Lamm, Allison Bishop, Betul Orcan, Darcy McGillicuddy, Ben Garcia, my teachers, staff and friends at Mathematics Department of University of Texas at Austin, my colleagues at Mathematics Department of Syiah Kuala University

and my friends Indonesian students and Ford fellows who have supported me during my hard time at UT.

I wish to thank everybody with whom I have shared experiences in life. From the people who first persuaded and got me interested into the study of mathematics, especially those who also played a significant role in my life, to those which with the gift of their company made my days more enjoyable and worth living.

Last but not the least, I would like to thank my family: my parents Farida Boerhan and Ramli Ibrahim, who always support and encourage me; my husband, Aidil Misra who sacrificing his happiness for the success of my study; my beloved sons Hatta and Hamka, who are the center of my life; my brothers and sisters, who always be there for me. To all of them I dedicate this report.

Contents

List of Tables

1. INTRODUCTION

Following the remarkable Claude Shannon's paper "A Mathematical Theory of Communication" [16] in 1948, coding theory has become a fast growth mathematical theory which has a wide area of application especially in communication system and information theory. Although recently codes over non-binary fields and even over rings have been studied and developed, in this paper we just focused on binary codes which have received the most attention since the beginning of coding theory.

Among all types of block codes, linear codes are the most studied. Because of their algebraic structure, they are easier to describe, encode and decode than nonlinear codes.

One class of codes that has many well known best error correcting codes are linear self-dual codes. One such code is the Reed-Muller RM(1,5) code that was used in the spacecraft Mariner 9 to send the gray image of Mars on 19 January 1972. Based on their structure, self-dual codes have a rich mathematical theory and strong connections with other areas of combinatorics, group theory and

lattice. In this paper we will discuss the basic properties and constructions of linear self-dual binary codes.

Definition 1.1. *Let $\mathbb{F}_2$ be the field on two elements. A* linear code *C of length n is a subspace of the vector space $\mathbb{F}_2^n$. If the dimension of C is k then C is called $[n,k]$-code over $\mathbb{F}_2$ and the 2^k elements of C are called* codewords. *The* generator matrix *of C is a $k \times n$ matrix whose rows are a basis of C. The dual code of C is defined to be $C^\perp = \{x \in \mathbb{F}_2^n \mid x \cdot y = \sum_{i=1}^n x_i y_i = 0$ for all $y \in C\}$. A code C is called* self-orthogonal *if $C^\perp \subseteq C$ and C is called* self-dual *if $C^\perp = C$.*

If C is an $[n,k]$ code then $C^\perp$ is an $[n,n-k]$-code, so that if C is a self-dual code then C is an $[n,n/2]$-code.

One important property of a code is its *minimum Hamming distance*, that is, the minimum number of distinct coordinates between any pair of distinct codewords. For a linear binary code, this is the same as *the minimum Hamming weight* of the code, which is the minimum number of ones in a nonzero codeword. It is important because if a code C has minimal Hamming weight d then C can correct $\lfloor (d-1)/2 \rfloor$ errors. That is, suppose that a codeword u is transmitted, but en route at most $\lfloor (d-1)/2 \rfloor$ of the bits are flipped, and the word received is w. Assuming that w differs from a codeword in at most $\lfloor (d-1)/2 \rfloor$ bits, and knowing

that the spheres of radius $\lfloor (d-1)/2 \rfloor$ about any two codewords are distinct, implies that the only possibility for the transmitted codeword is u. On the other hand, if more than $\lfloor (d-1)/2 \rfloor$ may occur during transmission, it may not be possible to uniquely identify the original codeword. Another parameter that is often optimized is the information rate of the code. The information rate of an $[n, k]$ code is defined to be k/n. It states that for every k bits of useful information, the code generates a total of n bits of data, of which $n - k$ are redundant. For self-dual codes, the transmission rate is always $1/2$. The closer the rate is to 1 the more efficient it is to encode information using the code. Here efficiency refers to the length of messages that are used to encode information.

A linear binary code of length n, dimension k, and minimum Hamming weight d is called an $[n, k, d]$-code, and a self-dual binary code of minimum Hamming weight d is an $[n, n/2, d]$-code. Note that in a self-dual code, each codeword is orthogonal to itself. Thus every codeword has an even number of ones and therefore the minimum Hamming weight of a self-dual code is even. In the rest of this report we will omit the adjective *Hamming* when discussing weights of codes.

One important goal in coding theory is to find the best code that can be applied for a specific function. A well-designed $[n, k, d]$

linear binary code is a code with three characteristics :

(1) The length n is small. Then each codeword is short and can be transmitted quickly.

(2) The number of codewords (2^k) is large. Then the information content of the code is large.

(3) The minimum distance d is large. Then the code can correct a large number of errors.

Of course satisfying all three characteristics is difficult. For example, if n is small then k and d will tend to be small as well. Therefore the main problem in algebraic coding theory is to fix one of the parameter (usually n), and optimize the values of the other two parameters.

The following are examples of self-dual binary codes. The first is the $[8, 4, 4]$ Hamming code h_8, one of whose generator matrices is G_1 where

$$G_1 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

It is easy to verify that all row vectors of this matrix are orthogonal to each other, so that all words spanned by those vectors will be orthogonal to each other. It means $h_8 = h_8^{\perp}$, and so h_8 is a

self-dual code with 16 codewords of length 8. The second example is the binary Golay code g_{24} with 12×24 generator matrix $G_2 = [I, B]$ where I is the 12×12 identity matrix and B is the matrix

$$B = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}.$$

It can be verified that $BB^T = I_{12}$ where B^T is the transpose of B. Indeed, using elementary row operations, every generator matrix of a self-dual code is equivalent to a matrix in the form $[I_{n/2}|B]$ where B is the $n/2 \times n/2$ matrix such that $BB^T = I_{n/2}$.

Gleason and Pierce [17] partition self-dual codes into two families: the family of self-dual codes in which there is at least one codeword with weight not divisible by 4 and the family in which the weight of each codeword is divisible by 4. A code in the

first family is called a *Type I* self-dual binary code or *singly-even*. A code in the second family is called a *Type II* or *doubly-even* self-dual code. Both of our previous examples of self-dual binary codes are Type II codes. One example of a Type I self-dual binary code is the *odd Golay code* h_{24}^{+} that is the $[24, 12, 6]$ code with the generator matrix

$$\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0
\end{bmatrix}.$$

Since the last row vector of the generator matrix G has weight 6 that is not divisible by 4, the code h_{24}^{+} contains at least a codeword with weight not divisible by 4. Thus it is a Type I code.

In the section 2 of this report we will discuss about the weight enumerator of a self-dual code which is the polynomial that records the number of codewords of each weight. One important result

on weight enumerators of linear codes is *MacWilliams' identity*, and we will prove it in section 2 for the special case of binary codes. Then, using MacWilliams' theorem we prove the fundamental *Gleason's theorem on self-dual codes*. Gleason's theorem states that the weight enumerator of a self-dual binary code is a sum of products of two specific polynomials. This restriction will help us determine the possible weight enumerators for the best self-dual codes of a given length.

When constructing self-dual codes, it is typical to fix the length and look for codes with the largest possible minimum weight. Using Gleason's theorem we can determine *the extremal weight enumerator* of a self-dual code, that is the weight enumerator of a given length that has the largest minimum weight. From this theorem we will determine the upper bound of a minimum weight of a self-dual code that is stated by the theorem below. This theorem is cited from the paper of C.L. Mallows and N. J. A. Sloane [13].

Theorem 1.1. *Let C be an $[n, n/2, d]$ self-dual binary code. Then*
(i) If C is type I, then

$$d \leq 2\lfloor n/8 \rfloor + 2 \tag{1.1}$$

(ii) If C is type II, then

$$d \leq 4\lfloor n/24 \rfloor + 4. \tag{1.2}$$

Mallows and Sloane proved this theorem by determining the explicit form for the extremal weight enumerators of self-dual binary codes using Bürmann's theorem. We will discuss this theorem more deeply in Section 3.

Codes of Type I satisfying $d \leq 2\lfloor n/8 \rfloor + 2$ only exist when $n = 2$, 4, 6, 8, 12, 14, 22, and 24. This fact is proved by H.N. Ward in 1976 in his paper [19], so the upper bound (i) is rarely sharp. In 1990 J.H. Conway and N.J.A. Sloane [4] strengthen this bound for an $[n, n/2, d]$-Type I self-dual binary code with $n \neq 2$, 12, 22, or 32 showing that $d \leq 2\lfloor n + 6 \rfloor /10$. Conway and Sloane proved this bound via a connection to the weight enumerator of the *shadow code* which is a particular translate of a self-dual code. We will discuss the shadow code in section 4. We also give examples of constructing weight enumerators of self-dual codes of length 38 and 40 based on Conway and Sloane's restriction on the weight enumerator of the shadow code. To compute these examples, we use the computer algebra system Mathematica.

Two binary codes are *equivalent* if they are the same up to a permutation of the coordinates of the codewords. The set of all permutations that preserve the code C forms a group that is called the *automorphism group Aut (C)* of the code which is the sub-group of the permutation group S_n where n is the length of the codewords of C.

As described in [14], there are several standard methods for constructing linear codes. The "gluing" method is a method that construct a new code from 2 existing codes by taking their direct sum. The length of the code will be the sum of the lengths of the two original codes. Another method of constructing a new code with larger length from an existing code is by adding a new coordinate in the codeword such that the sum of all coordinates will be 0. This method is called the extend method. The other method is opposite of the extend method and called the puncture method. In this method we delete some coordinates from the codewords of the original code. Of course the new code have smaller length. The shortening and extending method to construct a new code have many variations. We will describe one variant of the extend method which is known as the Gaborit-Otmani experimental method of constructing self-dual codes [6]. The method involves using different powers of orthogonal and permutation matrices to build a generator matrix for a new (hopefully better) self-dual code. The method will be explained in detail in Section 5.

2. GLEASON'S THEOREM

In general, a *weight enumerator* $W(x, y)$ is a homogeneous polynomial in two indeterminates x and y. In this section we will discuss the weight enumerator of a self-dual code and the constraints on weight enumerators of self-dual codes given by Gleason's theorem. The weight enumerator of a code is important because it generates the code's weight distribution. From the weight enumerator of a code C we can get information about how many codewords of certain weights exist in C and also the minimum weight of C.

Definition 2.1. *The weight enumerator $W = W(C)$ of a code C is the polynomial*

$$W(x, y) = \sum_{r=0}^{n} A_r x^{n-r} y^r \tag{2.1}$$

where A_r is the number of codewords in C with weight r.

If we take $x = 1$, we have a polynomial of a single variable y as the weight enumerator and still keep the information about the weight distribution of the code. The weight enumerator is defined as a function of two variables, however, because it is easier to describe the invariance properties of $W(x, y)$ than of $W(1, y)$.

Weight enumerators of self-dual codes have invariance properties that only depend on the type of the codes. The weight enumerator of a Type I self-dual code is invariant under a group of order 16 that is generated by the transformations $T_1((x,y)) = (x,-y)$ and $T_2((x,y)) = ((x+y)/\sqrt{2}, (x-y)/\sqrt{2})$. The weight enumerator of a Type II self-dual code is invariant under a group of order 192 that is generated by the transformations T_2 and $T_3((x,y)) = (x,iy)$.

In constructing self-dual codes, researchers use information given by the weight enumerator to design the generator matrix of the code. For example, suppose we want to construct an $[n,k,d]$ code with weight enumerator $W(x,y)$. Suppose that there are $[n_i,k_i,d_i]$ codes C_i for $i = 1,2$, such that $n = n_1 + n_2$, $k = k_1k_2$, and $d = min(d_1,d_2)$. Assume that G_1 and G_2 are the generator matrices of C_1 and C_2 respectively. If

$$W(x,y) = W_{C_1}(x,y)W_{C_2}(x,y)$$

then we can construct the generator matrix of the $[n,k,d]$-code by taking the block matrix

$$G = \begin{pmatrix} G_1 & \mathbf{0} \\ \mathbf{0} & G_2 \end{pmatrix}.$$

Therefore the new code is the direct sum $C_1 \oplus C_2$ consists of all vectors (u,v) where $u \in C_1$, and $v \in C_2$. Since

$$wt(u,v) = wt(u) + wt(v),$$

and

$$W_{C_1 \oplus C_2}(x, y) = W_{C_1}(x, y) W_{C_2}(x, y),$$

then $C_1 \oplus C_2$ is an $[n = n_1 + n_2, k = k_1 k_2, d = min(d_1, d_2)]$ code [12]. The process of construction could be reversed if we intend to construct a new code with smaller length. There are many other constructions of codes using different methods that use the information from the weight enumerators. Some of the constructions can be found in [2], [7], and [10].

In constructing self-dual codes of length n, the common purpose is to determine the *optimal self-dual code(s)* of length n with the largest minimum weight d. Equivalently, we can construct a weight enumerator that satisfies the equation (2.1) such that $A_0 = 1$, and $A_1 = A_2 = ... = A_{d-2} = 0$. Ideally, d is equal to the strongest upper bound mentioned in Theorem 1.1, and the resulting weight enumerator is called the *extremal weight enumerator*. If the self-dual code with this weight enumerator exists, then it is called the *extremal self-dual code* with length n. Of course an extremal self-dual code is also an optimal self-dual code but an optimal self-dual code may not be an extremal self-dual code. In many cases the extremal self-dual code of some lengths do not exist. For example, according to Theorem 1.1, if the extremal Type I self-dual code of length 38 exists, it would have minimum weight 10. By analyzing the weight enumerators of self-dual code

of length 38, we obtained that the extremal self-dual code of this length does not exist (the calculation can be seen in section 4). We found that the greatest possible minimum weight for Type I self-dual code of length 38 is 8. There are over 900 such codes [8]. Therefore, an optimal self-dual code of length 38 is the code with minimum weight 8 which is not an extremal self-dual code.

Let C be an $[n, k]$ linear code with weight enumerator

$$W(x, y) = \sum_{i=0}^{n} A_i x^{n-i} y^i$$

and let $C^{\perp}$ be its dual with weight enumerator

$$W^{\perp}(x, y) = \sum_{i=0}^{n} A_i^{\perp} x^{n-i} y^i.$$

Jessie MacWilliams in 1963 proved that the weight enumerator of a binary linear code and its dual are related by the MacWilliams identity which says that the weight enumerator of the dual $C^{\perp}$ is a linear transformation of the weight enumerator of C, as given in Theorem 2.1.

Theorem 2.1. *[MacWilliams' Identity] Let C be an [n, k] linear binary code with weight enumerator W(x,y). Then*

$$W^{\perp}(x,y) = \frac{1}{|C|} W(x+y, x-y). \tag{2.2}$$

MacWilliams' theorem is a key tool used to prove Gleason's theorem. Before proving MacWilliams' theorem, we must introduce the Hadamard transform.

Definition 2.2. *Let $f : \mathbb{F}_2^n \to \mathbb{C}$ be a mapping from the vector space $\mathbb{F}_2^n$ to the field of complex numbers. The* Hadamard transform of f, *denoted as $\hat{f}$, is defined to be*

$$\hat{f}(v) = \sum_{u \in \mathbb{F}_2^n} (-1)^{u \cdot v} f(u)$$

where $u \cdot v$ denotes the Euclidean inner product.

The proof of MacWilliams theorem depends on the following important lemma that relates the sum of a functional f over all codewords in $C^{\perp}$ and the sum of its Hadamard transformation over all codewords in C.

Lemma 2.1. *If C is an [n, k] linear binary code and $\hat{f}$ is the Hadamard transform of f, then*

$$\sum_{u \in C^{\perp}} f(u) = \frac{1}{|C|} \sum_{u \in C} \hat{f}(u) \tag{2.3}$$

Proof. By the definition of the Hadamard transform we have:

$$\sum_{u \in C} \hat{f}(u) = \sum_{u \in C} \sum_{v \in \mathbb{F}_2^n} (-1)^{u \cdot v} f(v)$$

$$= \sum_{v \in \mathbb{F}_2^n} f(v) \sum_{u \in C} (-1)^{u \cdot v}$$
$$= \sum_{v \in C^\perp} f(v) \sum_{u \in C} (-1)^{u \cdot v} + \sum_{v \in \mathbb{F}_2^n \setminus C^\perp} f(v) \sum_{u \in C} (-1)^{u \cdot v}$$

If $v \in C^\perp$, then $u \cdot v$ is always zero, so that

$$\sum_{u \in C} (-1)^{u \cdot v} = |C|.$$

For $v \in \mathbb{F}_2^n \setminus C^\perp$ let $\phi_v : C \to \mathbb{F}_2$ be the abelian group homomorphism such that $\phi_v(u) = u \cdot v$. Then $Ker(\phi_v) = \{u \in C \mid u \cdot v = 0\} \subsetneqq C$, and by the first fundamental isomorphism we obtain $C/Ker(\phi_v) \approx \mathbb{F}_2$. Applying Lagrange's theorem we have

$$|\{u \in C \mid u \cdot v = 0\}| = |\{u \in C \mid u \cdot v = 1\}|$$

that imply for $v \in \mathbb{F}_2^n \setminus C^\perp$,

$$\sum_{u \in C} (-1)^{u \cdot v} = (-1) \cdot |\{u \in C \mid u \cdot v = 0\}| + 1 \cdot |\{u \in C \mid u \cdot v = 1\}| = 0.$$

Thus we have

$$\sum_{u \in C} \hat{f}(u) = |C| \sum_{u \in C^\perp} f(v)$$

□

From now on, we will denote the weight of the codeword u by $wt(u)$. Now, we are ready to prove the MacWilliams' identity.

Proof of MacWilliams Identity. Consider the polynomial function $f : \mathbb{F}_2^n \to \mathbb{C}(x, y)$ such that

$$f(u) = x^{n - wt(u)} y^{wt(u)}.$$

Let $\hat{f}(u)$ be the Hadamard transformation of f where $u \cdot v$ is the Euclidean inner product of vectors u and v. Let $u = (u_1, \ldots, u_n)$ and $v = (v_1, \ldots, v_n)$, then

$$\hat{f}(u) = \sum_{v \in \mathbb{F}_2^n} (-1)^{u_1 v_1 + \cdots + u_n v_n} x^{n - wt(v)} y^{wt(v)}$$

$$= \sum_{v \in \mathbb{F}_2^n} (-1)^{u_1 v_1 + \cdots + u_n v_n} \prod_{i=1}^{n} x^{1 - v_i} y^{v_i}$$

$$= \sum_{v_1=0}^{1} \sum_{v_2=0}^{1} \cdots \sum_{v_n=0}^{1} \prod_{i=1}^{n} (-1)^{u_i v_i} x^{1 - v_i} y^{v_i}$$

$$= \prod_{i=1}^{n} \sum_{w=0}^{1} (-1)^{u_i w} x^{1-w} y^{w}$$

For $u_i = 0$ we have

$$\sum_{w=0}^{1} (-1)^{u_i w} x^{1-w} y^{w} = x + y$$

and for $u_i = 1$ we have

$$\sum_{w=0}^{1} (-1)^{u_i w} x^{1-w} y^{w} = x - y.$$

Thus we obtain

$$\hat{f}(u) = (x + y)^{n - wt(u)} (x - y)^{wt(u)}.$$

Then finally we get

$$\sum_{u \in C^{\perp}} x^{n - wt(u)} y^{wt(u)} = \frac{1}{|C|} \sum_{u \in C} (x + y)^{n - wt(u)} (x - y)^{wt(u)}.$$

□

Lemma 2.2. *Let $W(x,y) = \sum_{r=0}^{n} A_r x^{n-r} y^r$ be a formally self-dual weight enumerator over $\mathbb{F}_2$ which all weights are divisible by 2. Then $W(x,y) = W(y,x)$, and $A_{n-r} = A_r$ for $r = 0, 1, ..., n$.*

Proof. Since all weights are divisible by 2, we know $W(x,y) = W(x,-y)$. By MacWilliams' theorem,

$$W(x,y) = \frac{1}{|2^{n/2}|} W(x+y, x-y),$$

and since $W(x,y)$ is homogeneous of degree n we have

$$W(x,y) = W\left(\frac{x+y}{\sqrt{2}}, \frac{x-y}{\sqrt{2}}\right) \tag{2.4}$$

so that

$$\begin{aligned} W(x,y) &= W\left(\frac{x+y}{\sqrt{2}}, \frac{x-y}{\sqrt{2}}\right) = W\left(\frac{x+y}{\sqrt{2}}, \frac{y-x}{\sqrt{2}}\right), \\ &= W\left(\frac{(x+y)+(y-x)}{2}, \frac{(x+y)-(y-x)}{2}\right), \\ &= W(y,x). \end{aligned}$$

Thus $A_{n-r} = A_r$ for $r = 0, 1, ..., n$. □

Any polynomial $W(x,y)$ satisfying the equation (2.4) is called the *formal self-dual weight enumerator*, and it is called the *extremal formal self-dual weight enumerator* if the minimum weight is equal the bounds in Theorem 1.1.

The formal self-dual weight enumerator can be constructed even though a corresponding code may not exist. This is possible

because of Gleason's theorem on self-dual codes. The theorem states that the weight enumerator of a self-dual code is the sum of products of two specific polynomial functions. Gleason's theorem on self-dual code is the most important tool for determining the weight enumerators of self-dual codes. There are many papers written about this theorem and its generalization for self-dual codes over various finite fields. We will give the proof of the theorem based on the proof given by Berlekamp et. al. in [1].

Theorem 2.2. *Let C be an [n, n/2] self-dual code.*

(i) If C is a Type I code then the weight enumerator of C is of the form

$$W(x,y) = \sum_{i=0}^{\lfloor \frac{n}{8} \rfloor} a_i g^{n/2-4i} h^i, \tag{2.5}$$

where $g(x,y) = x^2 + y^2$ *and* $h(x,y) = x^2y^2(x^2 - y^2)^2$.

(ii)) If C is a Type II code then the weight enumerator of C is of the form

$$W(x,y) = \sum_{i=0}^{\lfloor \frac{n}{24} \rfloor} a_i g^{n/8-3i} h^i, \tag{2.6}$$

where $g(x,y) = y^8 + 14x^4y^4 + x^8$ *and* $h(x,y) = x^4y^4(x^4 - y^4)^4$.

Proof. Let the weight enumerator of C be

$$W(x,y) = \sum_{r=0}^{n} A_r x^{n-r} y^r$$

Since C is a self-dual binary code then by MacWilliams theorem we have

$$W(x,y) = W\left(\frac{x+y}{\sqrt{2}}, \frac{x-y}{\sqrt{2}}\right). \tag{2.7}$$

Let $z = y/x$ and $F(z) = \sum_{r=0}^{n} A_r z^r$. Using (2.7) write the MacWilliams identity for $F(z)$

$$\begin{aligned}\sum_{r=0}^{n} A_r z^r = W(1,z) &= W\left(\frac{1+z}{\sqrt{2}}, \frac{1-z}{\sqrt{2}}\right) \\ &= \sum_{r=0}^{n} A_r \left(\frac{1+z}{\sqrt{2}}\right)^{n-r} \left(\frac{1-z}{\sqrt{2}}\right)^{r}\end{aligned} \tag{2.8}$$

Now we prove item (i). If C is a type I code, then all codewords have even weight, so that we have

$$W(x,y) = W(x,-y). \tag{2.9}$$

From property (2.9) and the right-hand side of (2.8) if $\alpha \neq 1$ is the zero of $F(z)$ then $(-\alpha)$ and $(1-\alpha)/(1+\alpha)$ are also the zeroes of $F(z)$. Consider the transformation $\phi : \mathbb{C} \to \mathbb{C}$ such that $\phi(\alpha) = -\alpha$ and $\psi : \mathbb{C} \to \mathbb{C}$ such that $\psi(\alpha) = (1-\alpha)/(1+\alpha)$. Then we have $\phi^2 = 1$, $(\psi \circ \phi)^4 = 1$, and $\phi \circ (\psi \circ \phi) = (\psi \circ \phi)^3 \circ \phi$, so ϕ and ψ generate a group under composition, which is a dihedral group of order 8. Furthermore, since $F(\phi(z)) = F(z)$, and $F(\psi(z)) = F(z)$ then $F(z)$ is invariant under the dihedral group generated by ϕ and ψ. Thus, if $\alpha \neq 0, \pm 1$ is the zero of $F(z)$ then $\pm\alpha$, $\pm(1-\alpha)/(1+\alpha)$, $\pm(1+\alpha)/(1-\alpha)$ and $\pm 1/\alpha$ are also the

zeroes of $F(z)$.

There are two conditions that possibly occur:

(1) all the eight zeroes are distinct,

(2) some of the zeroes are equal.

In the next paragraph we will check both possibilities.

(1) If all of the eight zeroes of $F(z)$ are distinct then we will have

$$(z^2-\alpha^2)\left(z^2-\left(\frac{1}{\alpha}\right)^2\right)\left(z^2-\left(\frac{1-\alpha}{1+\alpha}\right)^2\right)\left(z^2-\left(\frac{1+\alpha}{1-\alpha}\right)^2\right)$$

as a factor of $F(z)$. By subtracting with $(1+z^2)^4$, we get the difference

$$-(1+\alpha^2)^4(\alpha)^{-2}(1-\alpha^2)^{-2}z^2(1-z^2)^2.$$

Thus we have

$$(z^2-\alpha^2)\left(z^2-\left(\frac{1}{\alpha}\right)^2\right)\left(z^2-\left(\frac{1-\alpha}{1+\alpha}\right)^2\right)\left(z^2-\left(\frac{1+\alpha}{1-\alpha}\right)^2\right)$$
$$=(1+z^2)^4+\beta z^2(1-z^2)^2 \qquad (2.10)$$

where $\beta=-(1+\alpha^2)^4(\alpha)^{-2}(1-\alpha^2)^{-2}$.

(2)If some of the zeroes are equal, we must consider several cases. If $\alpha=-1/\alpha$, then $\alpha=\pm i$, which implies that $(1+z^2)$ is a factor of $F(z)$. If $\alpha=\pm(1-\alpha)/(1+\alpha)$ then $\alpha=\pm1\pm\sqrt{2}$. Thus

$$((z^2-(1+\sqrt{2})^2)(z^2-(1-\sqrt{2})^2))^2=1-12z^2+38z^4-12z^6+z^8$$

is also the factor of $F(z)$. If we subtract this factor with $(1+z^2)^4$ then we have the difference is $-16z^2(1-z^2)^2$. In other words,

$$((z^2-(1+\sqrt{2})^2)(z^2-(1-\sqrt{2})^2))^2 = (1+z^2)^4 - 16z^2(1-z^2)^2$$

is a factor of $F(z)$.

Now we check for $\alpha = 0, \pm 1$. If $\alpha = 0$ then from the left-hand side of (2.8), $A_0 = 0$ and by lemma 2.2, $A_n = 0$. From the left-hand side of (2.8), $F(z)$ is divisible by z^2 and from the right-hand side, by $(1+z)^2(1-z)^2 = (1-z^2)^2$. Then $z^2(1-z^2)^2$ is a factor of $F(z)$. Similarly, if $\alpha = 1$ then from the right-hand side of (2.8), $A_n = 0$ and by lemma 2.2, $A_0 = 0$. From the left-hand side of (2.8), $F(z)$ is divisible by z^2 and from the right-hand side, by $(1+z)^2(1-z)^2 = (1-z^2)^2$. Then $z^2(1-z^2)^2$ is a factor of $F(z)$. If $\alpha = -1$, $F(z)$ also contains the factor $z^2(1-z^2)^2$.

We have got all factors of $F(z)$ for all possible zeroes. Now we can represent $F(z)$ as the product of the factors. Taking all factors of $F(z)$, we have

$$F(z) = a(1+z^2)^b[(1-z^2)^2z^2]^c \prod [(1+z^2)^4 + \beta z^2(1-z^2)^2],$$

where a is a complex number and b and c are nonnegative integers. By the relation $x^n\ F(z) = W(x,y)$, and $z = y/x$, we obtain

$$W(x,y) = a(x^2+y^2)^b[(x^2-y^2)^2x^2y^2]^c \prod [(x^2+y^2)^4+\beta x^2y^2(x^2-y^2)^2].$$

Expanding the product we get

$$W(x,y) = \sum_{r,s} K_{rs}(x^2+y^2)^r[(x^2-y^2)^2x^2y^2]^s$$

where $n = 2r + 8s$ and K_{rs} is a complex number. Thus

$$W(x,y) = \sum_{i=0}^{n/8} a_i(x^2+y^2)^{n/2-4i}[(x^2-y^2)^2x^2y^2]^i.$$

Now we prove item (ii). Recall that $F(z)$ is invariant under the transformation ϕ and ψ. If C is a Type II code, all weights are divisible by 4 so that $W(x,y) = W(x,iy)$. Then $F(z)$ is also invariant under transformation $\theta : \mathbb{C} \to \mathbb{C}$ such that $\theta(\alpha) = i\alpha$. Then we have $(\theta \circ \phi)^4 = 1$, $\psi^2 = 1$, and $(\theta \circ \phi \circ \psi)^3 = 1$. Thus transformation ϕ, ψ and θ under composition generate a symmetric group S_4 of order 24. Therefore, if $\alpha \neq 0, \pm 1, \pm i$ is a zero of $F(z)$ then $\varepsilon\alpha$, $\varepsilon 1/\alpha$, $\varepsilon(1-\alpha)/(1+\alpha)$, $\varepsilon(1+\alpha)/(1-\alpha)$, $\varepsilon(1-i\alpha)/(1+i\alpha)$, and $\varepsilon(1+i\alpha)/(1-i\alpha)$ are also the zeroes of $F(z)$ where $\varepsilon = \pm 1, \pm i$.

We check for two possibilities:

(1) all zeroes are distinct

(2) some zeroes are equal

Now we check both possibilities. (1)Now we analyze the case when the zeroes of $F(z)$ are distinct. In the proof of Type I code above we have shown that for eight distinct zeroes $\pm\alpha$, $\pm(1-$

$\alpha)/(1+\alpha)$, $\pm 1/\alpha$, and $\pm(1+\alpha)/(1-\alpha)$ we have the factor shown in (2.10). Rewrite this as

$$(1+14z^4+z^8)-\frac{1+14\alpha^4+\alpha^8}{\alpha^2(1-\alpha^2)^2}z^2(1-z^2)^2 = h-\frac{\gamma}{\alpha^2(1-\alpha^2)^2}g \tag{2.11}$$

where $h=(1+14z^4+z^8)$, $g=z^2(1-z^2)^2$, and $\gamma=1+14\alpha^4+\alpha^8$.

For the distinct zeroes $\pm i\alpha$, $\pm(1-i\alpha)/(1+i\alpha)$, $\pm 1/i\alpha$, and $\pm(1+i\alpha)/(1-i\alpha)$ we could subtitute α with $i\alpha$ in (2.11) and we get the factor

$$h+\left[\frac{\gamma}{\alpha^2(1+\alpha^2)^2}\right]g.$$

For the other distinct zeroes we apply $\alpha \to i(1-\alpha)/(1+\alpha)$ in (2.11) so that we have the factor

$$h+\left[\frac{4\gamma}{(1-\alpha^2)^2(1+\alpha^2)^2}\right]g$$

Multiplying all those factors we obtain the factor of $F(z)$:

$$\left(h-\frac{\gamma}{\alpha^2(1-\alpha^2)^2}g\right)\left(h+\left[\frac{\gamma}{\alpha^2(1+\alpha^2)^2}\right]g\right)\left(h+\left[\frac{4\gamma}{(1-\alpha^2)^2(1+\alpha^2)^2}\right]g\right)$$
$$= h^3+\frac{\gamma^3}{\alpha^4(1-\alpha^4)^4}(h+4g)g^2 = (1+14z^4+z^8)^3+\beta z^4(1-z^4)^4,$$

where $\beta=-(1+14\alpha^4+\alpha^8)^3\alpha^{-4}(1-\alpha^4)^{-4}$.

(2)Analyzing the multiple zeroes gave that $1+14z^4+z^8$ as the factor, for instance if

$$\alpha=\frac{1+i\alpha}{1-i\alpha}$$

then

$$\alpha = \frac{1}{2}(-(1+i) \pm (-1)^{1/4}\sqrt{6}).$$

If

$$\frac{1}{\alpha} = \frac{i(1+\alpha)}{1-\alpha}$$

then

$$\alpha = \frac{1}{2}((-1+i) \pm (-1)^{3/4}\sqrt{6}).$$

For

$$\alpha = \frac{1}{2}((-1+i) + (-1)^{3/4}\sqrt{6}),$$

then

$$\begin{aligned}\{&\varepsilon\alpha, \varepsilon 1/\alpha, \varepsilon(1-\alpha)/(1+\alpha), \varepsilon(1+\alpha)/(1-\alpha),\\ &\varepsilon(1-i\alpha)/(1+i\alpha), \varepsilon(1+i\alpha)/(1-i\alpha)\}. \qquad (2.12)\end{aligned}$$

are the zeroes of $F(z)$, where $\varepsilon = \pm 1, \pm i$.

Multiply all linear factors of $F(z)$ given by those zeroes, we have the following factor of $F(z)$

$$\begin{aligned}&\Big(z^2 - \frac{(1-i\alpha)^2}{(1+i\alpha)^2}\Big)\Big(z^2 - \frac{(1+i\alpha)^2}{(1-i\alpha)^2}\Big)\Big(z^2 - \frac{1}{\alpha^2}\Big)\Big(z^2 + \frac{1}{\alpha^2}\Big)\\ &(z^2-\alpha^2)(z^2+\alpha^2)\Big(z^2 + \frac{(-i+\alpha)^2}{(i+\alpha)^2}\Big)\Big(z^2 + \frac{(i+\alpha)^2}{(-i+\alpha)^2}\Big)\\ &\Big(z^2 - \frac{(1-\alpha)^2}{(1+\alpha)^2}\Big)\Big(z^2 + \frac{(1-\alpha)^2}{(1+\alpha)^2}\Big)\Big(z^2 - \frac{(1+\alpha)^2}{(1-\alpha)^2}\Big).\end{aligned}$$

Simplifying the above factor using Mathematica, we get the factor of $F(z)$

$$\prod_{j=1}^{24}(z-\alpha_j) = 1+42z^4+591z^8+2828z^{12}+591z^{16}+42z^{20}+z^{24}$$
$$= (1+14z^4+z^8)^3.$$

Here $\alpha_1, \ldots, \alpha_{24}$ are the 24 elements in the set

$$\{\varepsilon\alpha, \varepsilon 1/\alpha, \varepsilon(1-\alpha)/(1+\alpha), \varepsilon(1+\alpha)/(1-\alpha),$$
$$\varepsilon(1-i\alpha)/(1+i\alpha), \varepsilon(1+i\alpha)/(1-i\alpha)\}, \tag{2.13}$$

where $\varepsilon = \pm 1, \pm i$.

Analyzing other multiple zeroes also gives that $1+14z^4+z^8$ as a factor.

Now we check all factors of $F(z)$ if the zeroes of $F(z)$ is one of $0, 1$, or $\pm i$. Since $\phi(1) = -1$, $(\phi \circ \psi)(0) = -1$, $\theta(i) = -1$, and $(\theta \circ \phi)(-i) = -1$, and F is invariant under those transformations, then we have $F(-1) = 0$. From the right-hand side of (2.8) we have $A_0 = 0$, then by Lemma 2.2 $A_n = 0$. Thus from the left-hand side of (2.8) we get z^4 divides $F(z)$. From the right-hand side of (2.8) we have $(1-z)^4(1+z)^4 = (1-z^2)^4$ also divides $F(z)$. Since the code is a Type II we also have $F(z) = F(iz)$ that imply $(1-iz)^4(1+iz)^4 = (1+z^2)^4$ is also a factor of $F(z)$. Then we have $F(z)$ is divisible by $z^4(1-z^2)^4(1+z^2)^4 = z^4(1-z^4)^4$.

We have checked all factors of $F(z)$ for all possible zeroes. Multiply all factors will give the representation of $F(z)$. So,

$$F(z) = a(1+14z^4+z^8)^b[(1-z^4)^4z^4]^c \prod[(1+14z^4+z^8)^3+\beta z^4(1-z^4)^4],$$

where a is a complex number and b, and c are nonnegative integers. Applying $x^nF(z) = W(x,y)$ and $z = y/x$, we obtain

$$\begin{aligned} W(x,y) &= a(x^8+14x^4y^4+y^8)^b[(x^4-y^4)^4x^4y^4]^c \\ &\quad \prod[(x^8+14x^4y^4+y^8)^3+\beta x^4y^4(x^4-y^4)^4]. \end{aligned}$$

Expanding the product we get

$$W(x,y) = \sum_{r,s} K_{rs}(x^8+14x^4y^4+y^8)^r[x^4y^4(x^4-y^4)^4]^s,$$

where $n = 8r+24s$, and K_{rs} is a complex number. Thus we have

$$W(x,y) = \sum_{i=0}^{n/24} a_i(x^8+14x^4y^4+y^8)^{n/8-3i}[x^4y^4(x^4-y^4)^4]^i.$$

□

3. UPPER BOUNDS ON MINIMUM WEIGHTS

Mallows and Sloane [13] showed that the minimum weight of a Type I self-dual code of length n is at most $2\lfloor n/8 \rfloor + 2$, while the minimum weight of a Type II self-dual code of length n is at most $4\lfloor n/24 \rfloor + 4$ as given in Theorem 1.1. In determining these bounds, they gave an explicit formula for the extremal weight enumerator as given in Theorem 3.2, which is an extension of Gleason's theorem from section 2. In proving Theorem 3.2, we use the Bürmann-Lagrange's theorem which is given without proof. The proof can be found in [20]. The proof of Theorem 1.1 follows directly from Theorem 3.2 that allows us to find the coefficients of the extremal weight enumerator.

Theorem 3.1. *[Bürmann-Lagrange's Theorem] Let $f(z)$ and $\Phi(z)$ be analytic functions near $x = 0$, with $\Phi(0) \neq 0$. Provided that the equation $z = \epsilon\Phi(z)$ defines z uniquely in some neighborhood of the origin, then $f(z)$ can be expanded in powers of ϵ as follows:*

$$f(z) = f(0) + \sum_{m=1} \frac{[\epsilon]^m}{m!} \frac{d^{m-1}}{da^{m-1}} [f'(a)(\Phi(a))^m]_{a=0}$$

valid for sufficiently small ϵ.

Now we will prove the following theorem about the extremal

weight enumerator of self-dual binary codes.

Theorem 3.2. *The extremal weight enumerator of self-dual binary code of length n is given by:*

(i) If all weights are divisible by 2, then

$$W(y) = \sum_{k=0}^{\lfloor n/8 \rfloor} a_k (1+y^2)^{n/2-4k} (y^2(1-y^2)^2)^k,$$

where $a_0 = 1$ and $a_k, 1 \leq k \leq \lfloor n/8 \rfloor$, is equal to

$$\frac{n}{2k} \sum_{r=0}^{k-1} (-1)^{r+1} \binom{n/2-4k+r}{r} \binom{3k-r-2}{k-r-1}$$

(ii) If all weights are divisible by 4, then

$$W(y) = \sum_{k=0}^{\lfloor n/24 \rfloor} a_k (1+14y^4+y^8)^{n/8-3k} (y^4(1-y^4)^4)^k,$$

where $\alpha_0 = 1$ and $\alpha_k, 1 \leq k \leq \lfloor n/24 \rfloor$, is equal to

$$\frac{n}{8k} \sum_{r=0}^{k-1} (r+1) \binom{5k-r-2}{k-r-1} \sum_{i=0}^{\lfloor (r+1)/2 \rfloor} \frac{(-1)^i (-14)^{r+1-2i} ((n/8)-3k+r-i)!}{((n/8)-3k)!(r+1-2i)!i!}$$

Proof. For a type I code, choose a_k so that

$$W(y) = \sum_{k=0}^{\lfloor n/8 \rfloor} a_k f^{\frac{n}{2}-4k} g^k = 1 + \sum_{k=\lfloor n/8 \rfloor + 1}^{n/2} A_{2k} y^{2k} \tag{3.1}$$

where $f(y) = 1+y^2$ and $g(y) = y^2(1-y^2)^2$. We can choose a_k iteratively so that all coefficients of y^{2i} are zero for $i \leq \lfloor n/8 \rfloor$.

Dividing by $f^{-(n/2)}$ and substituting z for y^2 we have

$$[f(z)]^{-(\frac{n}{2})} = \sum_{k=0}^{\lfloor n/8 \rfloor} a_k [f(z)]^{-4k} [g(z)]^k - [f(z)]^{-n/2} \sum_{k=\lfloor n/8 \rfloor + 1}^{n/2} A_{2k} z^k, \tag{3.2}$$

where

$$\frac{g(z)}{[f(z)]^4} \;=\; \frac{z(1-z)^2}{(1+z)^4} \;\sim\; z \text{ for } z \text{ small.}$$

By the Bürmann-Lagrange's Theorem in power of $\epsilon = g(z)/([f(z]^4)$, we expand $[f(z)]^{-(\frac{n}{2})}$ with

$$\Phi(z) \;=\; \frac{z}{\epsilon} \;=\; \frac{[f(z)]^4}{(1-z)^2}.$$

Thus

$$[f(z)]^{-(\frac{n}{2})} = \sum_{k=0}^{\infty} \alpha_k \epsilon^k, \tag{3.3}$$

where $\alpha_0 = 1$ and

$$\begin{aligned}
\alpha_k &= \frac{1}{k!} \left[\frac{d^{k-1}}{dz^{k-1}} \frac{df^{-(\frac{n}{2})}}{dz} \left(\frac{z}{\epsilon}\right)^k \right]_{z=0} \\
&= -\frac{n}{2(k!)} \left[\frac{d^{k-1}}{dz^{k-1}} [f(z)]^{-\frac{n}{2}-1} f'(z) \frac{f^{4k}}{(1-z)^{2k}} \right]_{z=0} \\
&= -\frac{n}{2(k!)} \left[\frac{d^{k-1}}{dz^{k-1}} f' f^{-(\frac{n}{2}-4k+1)} (1-z)^{-2k} \right]_{z=0}
\end{aligned}$$

By the Product Rule, we get

$$\alpha_k = -\frac{n}{2(k!)} \left[\sum_{r=0}^{k-1} \binom{k-1}{r} \frac{d^r}{dz^r} f' f^{-(\frac{n}{2}-4k+1)} \frac{d^{k-1-r}}{dz^{k-1-r}} (1-z)^{-2k} \right]_{z=0},$$

which is equivalent to

$$\alpha_k = \frac{n}{2(k!)(\frac{n}{2}-4k)}\left[\sum_{r=0}^{k-1}\binom{k-1}{r}\frac{d^{r+1}}{dz^{r+1}}f^{-(\frac{n}{2}-4k)}\frac{d^{k-1-r}}{dz^{k-1-r}}(1-z)^{-2k}\right]_{z=0}. \tag{3.4}$$

Using the chain rule, for $s > 0$, we have

$$\left[\frac{d^r}{dz^r}(1+\alpha z)^{-s}\right]_{z=0} = \frac{(s-1+r)!}{(s-1)!}(-\alpha)^r.$$

Thus we obtain for $k = 0, 1, \cdots, \lfloor n/8 \rfloor$,

$$\left[\frac{d^{r+1}}{dz^{r+1}}(f)^{-(\frac{n}{2}-4k)}\right]_{z=0} = \frac{(\frac{n}{2}-4k+r)!}{(\frac{n}{2}-4k-1)!}(-1)^{r+1}, \tag{3.5}$$

and

$$\left[\frac{d^{k-r-1}}{dz^{k-r-1}}(1-z)^{-2k}\right]_{z=0} = \frac{(3k-2-r)!}{(2k-1)!}. \tag{3.6}$$

By substituting (3.5) and (3.6) into (3.4), we have that α_k is equal to

$$\frac{n}{2(k!)(\frac{n}{2}-4k)}\left[\sum_{r=0}^{k-1}\binom{k-1}{r}\frac{(\frac{n}{2}-4k+r)!}{(\frac{n}{2}-4k-1)!}(-1)^{r+1}\frac{(3k-2-r)!}{(2k-1)!}\right]_{z=0}. \tag{3.7}$$

Comparing (3.2) and (3.3) we see that

$$a_k = \alpha_k \text{ for } k = 0, 1, \ldots, \lfloor n/8 \rfloor,$$

and

$$\sum_{k=\lfloor n/8 \rfloor+1}^{\infty} \alpha_k [f(z)]^{n/2-4k}[g(z)]^k \;=\; -\sum_{k=\lfloor n/8 \rfloor+1}^{n/2} A_{2k} z^k. \tag{3.8}$$

Thus for $1 \leq k \leq \lfloor n/8 \rfloor$, equation (3.7) simplifies to

$$a_k = \alpha_k = \frac{n}{2k} \sum_{r=0}^{k-1} (-1)^{r+1} \binom{n/2 - 4k + r}{r} \binom{3k - r - 2}{k - r - 1}.$$

For a type II code, choose a_k so that

$$W(y) = \sum_{k=0}^{\lfloor n/24 \rfloor} f^{\frac{n}{8}-3k} g^k = 1 + \sum_{\lfloor n/24 \rfloor + 1}^{n/4} A_{4k} y^{4k}$$

where $f(y) = 1 + 14y^4 + y^8$ and $g(y) = y^4(1 - y^4)^4$. Dividing by $f^{-(n/8)}$ and substituting z for y^4 we have

$$[f(z)]^{-\frac{n}{8}} = \sum_{k=0}^{\lfloor n/24 \rfloor} a_k [f(z)]^{-3k} [g(z)]^k - [f(z)]^{-\frac{n}{8}} \sum_{\lfloor n/24 \rfloor + 1}^{n/4} A_{4k} z^k, \quad (3.9)$$

where

$$\frac{g(z)}{[f(z)]^3} \;=\; \frac{z(1-z)^4}{(1 + 14z + z^2)^3} \;\sim\; z \text{ for } z \text{ small.}$$

By the Bürmann-Lagrange's Theorem in power of $\epsilon = g(z)/([f(z]^3)$, we expand $[f(z)]^{-(\frac{n}{8})}$ with

$$\Phi(z) \;=\; \frac{z}{\epsilon} \;=\; \frac{[f(z)]^3}{(1-z)^4}.$$

Thus

$$[f(z)]^{-(\frac{n}{8})} = \sum_{k=0}^{\infty} \alpha_k \epsilon^k, \quad (3.10)$$

where

$$\alpha_k = \frac{1}{k!}\left[\frac{d^{k-1}}{dz^{k-1}}\frac{df^{-(\frac{n}{8})}}{dz}\left(\frac{z}{\phi}\right)^k\right]_{z=0}$$
$$= -\frac{n}{8k!}\left[\frac{d^{k-1}}{dz^{k-1}}f^{-\frac{n}{8}-1}f'\frac{f^{3k}}{(1-z)^{4k}}\right]_{z=0}$$
$$= -\frac{n}{8k!}\left[\frac{d^{k-1}}{dz^{k-1}}f'f^{-(\frac{n}{8}-3k+1)}(1-z)^{-4k}\right]_{z=0}.$$

By the Product Rule we have,

$$\alpha_k = -\frac{n}{8k!}\left[\sum_{r=0}^{k-1}\binom{k-1}{r}\frac{d^r}{dz^r}f'f^{-(\frac{n}{8}-3k+1)}\frac{d^{k-1-r}}{dz^{k-1-r}}(1-z)^{-4k}\right]_{z=0},$$

which is equivalent to

$$\alpha_k = \frac{n}{8(k)!(\frac{n}{8}-3k)}\left[\sum_{r=0}^{k-1}\binom{k-1}{r}\frac{d^{r+1}}{dz^{r+1}}f^{-(\frac{n}{8}-3k)}\frac{d^{k-1-r}}{dz^{k-1\frac{r}{2}-r}}(1-z)^{-4k}\right]_{z=0} \tag{3.11}$$

We can simplify the expression of α_k using the chain rule; for $s > 0$

$$\left[\frac{d^r}{dz^r}(1+\alpha z)^{-s}\right]_{z=0} = \frac{(s-1+r)!}{(s-1)!}(-\alpha)^r$$

For $D = d/dt$, Faá Di Bruno's formula (see [9]) states that for functions f and g such that $D^k(f(t))$, and $D^r(g(t))$ exist for all k and r, we have

$$D^r(f(g(t))) = \sum \frac{r!}{k_1!\ldots k_r!}D^k(f(g(t)))\left[\frac{D(g(t))}{1!}\right]^{k_1}\cdots\left[\frac{D^r(g(t))}{r!}\right]^{k_r},$$

where the sum is over all weak compositions of r of length r. (A weak composition of r of length r is a tuple of non-negative integers $k_1, \ldots, k_r$ such that $k_1+2k_2+\cdots+rk_r = r$, and $k = k_1+\cdots+k_r$).

Using Faá Di Bruno's formula for the function $(1 + \alpha z + z^2)^{-s}$ where $s > 0$, and from the fact that the degree of $g(z)$ is 2, then $\left[\frac{(d/dz)^j(g(z))}{j!}\right]^{k_j} = 0$ for every $j \geq 3$. Thus $\left[\frac{(d/dz)^j(g(z))}{j!}\right]^{k_j}$ is non-zero only for $j = 1$, or $j = 2$. If $k_2 = i$ then $k_1 = r - 2i$ and $k = r - i$, for every $i = 0, \ldots, \lfloor r/2 \rfloor$. Then, for $s > 0$ we have

$$\left[\frac{d^r}{dz^r}(1 + \alpha z + z^2)^{-s}\right]_{z=0} = \left[\sum_{i=0}^{\lfloor r/2 \rfloor} \frac{r!}{(r-2i)!(i)!}\, \gamma\right]_{z=0},$$

where

$$\gamma = \frac{d^{r-i}}{dt^{r-i}}(1+\alpha z+z^2)^{-s}\left(\frac{d}{dt}(1 + \alpha z + z^2)\right)^{r-2i}\left(\frac{1}{2!}\frac{d^2}{dt^2}(1 + \alpha z + z^2)\right)^{i}.$$

Thus we have,

$$\left[\frac{d^r}{dz^r}(1 + \alpha z + z^2)^{-s}\right]_{z=0} = \sum_{i=0}^{\lfloor r/2 \rfloor} \frac{r!(s + r - 1 - i)!(-\alpha)^{r-2i}(-1)^i}{(s-1)!(r-2i)!i!}.$$

Then, for $k = 0, 1, \ldots, \lfloor n/24 \rfloor$,

$$\left[\frac{d^{r+1}}{dz^{r+1}}(f)^{-(n/8-3k)}\right]_{z=0} \text{ is equal to}$$

$$\sum_{i=0}^{\lfloor (r+1)/(2) \rfloor} \frac{(r+1)!(n/8-3k+r-i)!(-\alpha)^{r+1-2i}(-1)^i}{(n/8-3k-1)!(r+1-2i)!i!},$$

and

$$\left[\frac{d^{k-r-1}}{dz^{k-r-1}}(1-z)^{-4k}\right]_{z=0} = \frac{(5k-2-r)!}{(4k-1)!}.$$

Then we have for $k = 0, 1, \ldots, \lfloor n/24 \rfloor$,

$$\alpha_k = \frac{n}{8(k)!(n/8-3k)}\left[\sum_{r=0}^{k-1}\binom{k-1}{r}\frac{(5k-2-r)!}{(4k-1)!}\,\gamma\right]$$

where

$$\gamma = \sum_{i=0}^{\lfloor r/2 \rfloor}\frac{(r+1)!(n/8-3k+r-i)!(-\alpha)^{r+1-2i}(-1)^i}{(n/8-3k-1)!(r+1-2i)!i!}$$

Comparing (3.9) and (3.10) we see that

$$a_k = \alpha_k \text{ for } k = 0, 1, \ldots, \lfloor n/24 \rfloor,$$

and

$$\sum_{k=\lfloor n/24 \rfloor+1}^{\infty}\alpha_k[f(z)]^{n/8-3k}[g(z)]^k \;=\; -\sum_{k=\lfloor n/24 \rfloor+1}^{n/8}A_{4k}z^k. \quad (3.12)$$

Finally we get the following expression: for $1 \le k \le n/24$,

$$a_k = \alpha_k = \frac{n}{8k}\sum_{r=0}^{k-1}(r+1)\binom{5k-r-2}{k-r-1}\gamma$$

where

$$\gamma = \sum_{i=0}^{\lfloor (r+1)/2 \rfloor}\frac{(-1)^i(-14)^{r+1-2i}((n/8)-3k+r-i)!}{((n/8)-3k)!(r+1-2i)!i!}$$

□

Since we have determined all coefficients of the extremal weight enumerator for both types of self-dual codes, the next important information that we could gain from the formula is the number of codewords of minimum nonzero weight in the extremal weight enumerator. The following theorem will give the formula for those numbers. Theorem 1.1 will then follow from Theorem 3.3.

Theorem 3.3. *The number of codewords of minimum nonzero weight in the extremal weight enumerator are given below.*

(i) For for $m = \lfloor n/8 \rfloor$, $A_{2(m+1)}$ *is equal to*

$$\frac{4m}{m+1}\Big(3\binom{3m}{m-1}+\binom{3m-2}{m-3}+3\binom{3m-1}{m-2}+\binom{3m+1}{m}\Big), \quad \text{if } n = 8m;$$

$$\frac{4m+1}{m+1}\Big(2\binom{3m}{m-1}+\binom{3m-1}{m-2}+\binom{3m+1}{m}\Big), \quad \text{if } n = 8m+2;$$

$$\frac{4m+2}{m+1}\Big(\binom{3m}{m-1}+\binom{3m+1}{m}\Big), \quad \text{if } n = 8m+4;$$

$$\frac{3+4m}{m+1}\binom{3m+1}{m}, \quad \text{if } n = 8m+6;$$

(ii) For type II code for $m = \lfloor n/24 \rfloor$, $A_{4(m+1)}$ *is equal to*

$$\binom{n}{5}\binom{5m-2}{m-1}\Big/\binom{4m+4}{5}, \quad \text{if } n = 24m;$$

$$\frac{1}{4}n(n-1)(n-2)(n-4)\frac{(5m)!}{m!(4m+4)!}, \quad \text{if } n = 24m+8;$$

$$\frac{3}{2}n(n-2)\frac{(5m+2)!}{m!(4m+4)!}, \quad \text{if } n = 24m+16.$$

The next proof will simultaneously prove Theorem 3.3 and Theorem 1.1.

Proof of Theorem 1.1. (i) We can obtain $A_{2(m+1)}$ from the equation (3.8) by expanding the left hand side in powers of z. In particular

$$A_{2(m+1)} = -\alpha_{m+1},$$

then using (3.4) we have

$$\alpha_{m+1} = \frac{n}{2(m+1)!(4m+4-n/2)}\,\gamma$$

where

$$\gamma = \left[\sum_{r=0}^{m} \binom{m}{r} \frac{d^{r+1}}{dz^{r+1}} f^{-(\frac{n}{2}-4(m+1))} \frac{d^{m-r}}{dz^{m-r}} (1-z)^{-2(m+1)}\right]_{z=0}$$

For $n = 8m$ we have

$$\left[\frac{d^{r+1}}{dz^{r+1}}(f)^{-(\frac{n}{2}-4(m+1))}\right]_{z=0} = \frac{4!}{(4-r-1)!} \text{ for } r \leq 3,$$

and

$$\left[\frac{d^{r+1}}{dz^{r+1}}(f)^{-(\frac{n}{2}-4(m+1))}\right]_{z=0} = 0 \text{ for } r \geq 4.$$

We also have

$$\left[\frac{d^{m-r}}{dz^{m-r}}(1-z)^{-2(m+1)}\right]_{z=0} = \frac{(3m+1-r)!}{(2m+1)!},$$

so that we have

$$A_{2(m+1)} = \frac{m}{(m+1)} \sum_{r=0}^{3} (r+1) \binom{4}{r+1} \binom{3m-r+1}{m-r}.$$

The rest of the theorem follows by substituting $n = 8m+2,\ 8m+4,$ or $8m+6$ as appropriate.

(ii) The proof of Type II codes is similar to the proof of Type I codes. We have

$$A_{4(m+1)} = -\alpha_{m+1},$$

and from (3.11) we have

$$\alpha_{m+1} = -\frac{n}{8(m+1)!(3(m+1)-\frac{n}{8})}\,\gamma$$

where

$$\gamma = \left[\sum_{r=0}^{m}\binom{m}{r}\frac{d^{r+1}}{dz^{r+1}}f^{-(\frac{n}{8}-3(m+1))}\frac{d^{m-r}}{dz^{m-r}}(1-z)^{-4(m+1)}\right]_{z=0}.$$

For $n = 24m$ we have

$$\left[\frac{d^{m-r}}{dz^{m-r}}(1-z)^{-4(m+1)}\right]_{z=0} = \frac{(5m+3-r)!}{(4m+3)!}.$$

Using Faá Di Bruno's formula we get

$$\left[\frac{d^{r+1}}{dz^{r+1}}(1+14z+z^2)^3\right]_{z=0} = \sum_{i=0}^{\lfloor\frac{r+1}{2}\rfloor}\frac{(r+1)!\,(3)!\,(14)^{r+1-2i}}{(3-r-1+i)!\,(r+1-2i)!\,i!},$$

for $r \leq 2$, and

$$\left[\frac{d^{r+1}}{dz^{r+1}}(1+\alpha z+z^2)^3\right]_{z=0} = 0$$

for $r \geq 3$.

Then we have that $A_{4(m+1)}$ is equal to

$$\frac{m}{3(m+1)}\left[\sum_{r=0}^{2}(r+1)\binom{5m+3-r}{m-r}\sum_{i=0}^{\lfloor\frac{r+1}{2}\rfloor}\frac{(3)!\,(14)^{r+1-2i}}{(3-r-1+i)!\,(r+1-2i)!\,i!}\right].$$

Simplifying the above expression we have for $n = 24m$

$$A_{4(m+1)} = \binom{n}{5}\binom{5m-2}{m-1}\Big/\binom{4m+4}{5}.$$

The rest of the theorem follows by substituting $n = 24m + 8$, or $24m + 16$ as appropriate.

We have shown that the number $A_{2(\lfloor n/8 \rfloor+1)}$ and $A_{4(\lfloor n/24 \rfloor+1)}$ in the extremal weight enumerator of Type I and Type II self-dual code are never zero. This implies the following conclusion. Let C be an $[n, n/2, d]$ self-dual binary code. Then

(i) If C is type I, then

$$d \leq 2\lfloor n/8 \rfloor + 2.$$

(ii) If C is type II, then

$$d \leq 4\lfloor n/24 \rfloor + 4.$$

□

4. THE SHADOW CODE

In this section we will introduce the definition of a *shadow code* which is a particular translate of a linear binary code and discuss the restrictions of the weight enumerator of the shadow code that is given by Conway and Sloane on [4]. Then we will specify those restrictions for a shadow code of a self-dual code that will be useful in determining the weight enumerator of the self-dual code. In the end of this section we give some examples in determining the weight enumerators of self-dual codes based on the Gleason's theorem and the restriction of the weight enumerator of the shadow code for self-dual codes of length 38 and 40. All computations were done using Mathematica.

Definition 4.1. *Let C be a binary linear $[n, k, d]$ self-orthogonal code. Let $C^{(0)}$ be the subcode of $C^{\perp}$ containing all codewords with weights divisible by 4. The shadow code $S = S(C)$ consists of all vectors u such that $u \cdot v = 0$ for all $v \in C^{(0)}$ and $u \cdot v = 1$ for all $v \in C^{\perp} \setminus C^{(0)}$. The vectors in S are called the* parity vectors for $C^{\perp}$.

The weight enumerator of the shadow code S of code C are deter-

mined by considering the weight enumerator of C. The following theorem will give some properties of the shadow code and its weight enumerator. We use some basic algebra manipulation to prove the theorem.

Theorem 4.1. *Let C be an [n, k, d]-self-orthogonal code and let $S = S(C)$ be its shadow. If the weight of each vector in $C^{\perp}$ are divisible by 4 then $S = C$. If not then:*

(a) S is a nonlinear code, given by

$$S = C^{(0)\perp} \setminus C. \tag{4.1}$$

(b) If $u, v \in S$ then $u + v \in C$.

(c) Let $S(x, y) = \sum B_r x^{n-r} y^r$ be the weight enumerator of S. Then B_r are nonnegative integers satisfying $B_r = B_{n-r}$ for all r,

$$B_0 = 0, \tag{4.2}$$

$$B_r \leq 1 \qquad \text{for } r < d/2, \tag{4.3}$$

$$B_{d/2} \leq 2n/d, S = C^{(0)\perp} \setminus C. \tag{4.4}$$

$$\text{at most one } B_r \text{is nonzero for } r < (d+1)/2. \tag{4.5}$$

$$B_r \leq A(n, d, r)^1, \qquad \text{for all } r \tag{4.6}$$

1

[1] A(n, d, r) denotes the maximal possible number of binary vectors of length n, weight r, and Hamming distance at least d apart [11].

(d) If W(x, y) and $W^{\perp}$(x,y) are the weight enumerators of C and $C^{\perp}$ respectively then

$$S(x,y) = W\left(\frac{(1+i)x+(1-i)y}{2}, \frac{(1-i)x+(1+i)y}{2}\right) \tag{4.7}$$

and

$$S(x,y) = \frac{1}{2^{n-k}} W^{\perp}(x+y, i(x-y)) \tag{4.8}$$

Proof. First we will show that the shadow code $S = C$ or $S = C^{(0)\perp} \setminus C$. Let $u \in S$. Consider the linear transformation $\phi_u : C^{\perp} \to \mathbb{F}_2$ such that $\phi_u(v) = u \cdot v$. By the definition of S,

$$Ker(\phi_u) = C^{(0)}.$$

If all weights in $C^{\perp}$ are divisible by 4 then

$$Ker(\phi_u) = C^{(0)} = C^{\perp}$$

and $S = C$. Otherwise, by the first fundamental theorem of isomorphism we get that $C^{(0)}$ is a subcode of $C^{\perp}$ of index 2, and

$$C^{(0)\perp} = C \cup (a + C), \text{ for some } a \notin C.$$

We will show that $S = a + C$. By the definition of S, if $u \in S$ then $u \cdot v = 0$ for every $v \in C^{(0)}$, so that $S \subset C^{(0)\perp}$. We also have $u \cdot v = 1$ for every $v \in C^{\perp} \setminus C^{(0)}$, so that $u \notin C$. Then, S is not contained in C. Thus we have $S \subseteq C^{(0)\perp} \setminus C$. Conversely, if $u \in C^{(0)\perp} \setminus C$ then for every $w \in C^{(0)}$ we have

$w \cdot u = 0$, and for every $v \notin C^{(0)}$ we have $u \cdot v = 1$. Therefore, if $v \in C^{\perp} \setminus C^{(0)}$ we have $v \cdot u = 1$. Any $v' \in C^{\perp} \setminus C^{(0)}$ can be written as $v' = v + w$, for some $w \in C^{(0)}$. This implies that $w \cdot u = 0$, showing that $v' \cdot u = v \cdot u + w \cdot u = 1$. Then we can conclude that if $u \in C^{(0)\perp} \setminus C$ then for every $v \in C^{(0)}$ we have $u \cdot v = 0$, and for every $v' \in C^{\perp} \setminus C^{(0)}$ we have $v' \cdot u = 1$. By the definition of S we hav! e $u \in S$. This prove (a) i.e., $S = C^{(0)\perp} \setminus C = a + C$.

Next we will prove (b), i.e., if $u, v \in$ S then u + v $\in$ C. If $S = C$, then it is clear (b) is satisfied. If $S = C^{(0)\perp} \setminus C$, then for $u, v \in S$, and for $w \in C^{\perp}$ we have

$$u \cdot w = 0 \text{ and } v \cdot w = 0, \text{ if } w \in C^{(0)}.$$

Otherwise,

$$u \cdot w = 1 \text{ and } v \cdot w = 1, \text{ if } w \in C^{\perp} \setminus C^{(0)}.$$

Therefore, we get

$$(u + v) \cdot w = u \cdot w + v \cdot w = 0, \text{ for every } w \in C^{\perp},$$

that imply $u + v \in C$.

Now, we will prove that the number of codewords of weight k in the shadow code is the same as the number of codewords of weight $n - k$. We prove this by using the similar property of the self-dual code. Since every codeword in a self-dual code has even

weight, this implies that all-ones vector, written $\mathbf{1}$ is orthogonal to all vectors in C and since C is self-dual then $\mathbf{1} \in C$. Therefore, if $u \in C$ with $wt(u) = k$ then $\mathbf{1} + u \in C$, with

$$wt(\mathbf{1} + u) = wt(\mathbf{1}) - wt(u) = n - k.$$

Thus the number of codewords of weight k is the same as the number of codewords of weight $n - k$. This gives $W(y, x) = W(x, y)$.

To prove that $S(y, x) = S(x, y)$ we need to use the relation between $S(x, y)$ and $W(x, y)$. Assuming that (4.7) is true, then we have

$$S(y, x) = W\left(\frac{(1+i)y + (1-i)x}{2}, \frac{(1-i)y + (1+i)x}{2}\right)$$
$$= W\left(\frac{(1-i)y + (1+i)x}{2}, \frac{(1+i)y + (1-i)x}{2}\right) = S(x, y),$$

hence $B_r = B_{n-r}$ for all r.

Equation (4.2) is satisfied because $\mathbf{0}$ is not a parity vector. Equation (4.3) holds because if $B_r > 1$ for $r < d/2$ there will exist at least two distinct vectors u, v such that $wt(u) < d/2$ and $wt(v) < d/2$. Then by (b) $u + v \in C$ with $wt(u + v) < d$, a contradiction. Thus $B_r \leq 1$ for $r < d/2$.

Equation (4.6) also holds by (b) and a similar argument, if $B_r > A(n, d, r)$ and since $A(n, d, r)$ denotes the maximal possible number of binary vectors of length n, weight r, and distance at least d

apart, then there are vectors $u, v \in S$ with weight r with distance less than d, by (b) $u + v \in C$ but $wt(u + v) < d$, a contradiction. Equation(4.5) also holds from (b) by similar argument. Equation (4.4) is a special case of (4.6).

The next step is showing the equation for $S(x, y)$ in terms of W and $W^{\perp}$ given in (d). To prove (d) we compute the weight enumerators of $C^{(0)\perp}$ and subtract from it the weight enumerator of C as below. By MacWilliams identity if $W(x, y)$ is the weight enumerator of C, then the weight enumerator of $C^{\perp}$ is

$$W^{\perp}(x, y) = W_{C^{\perp}}(x, y) = \frac{1}{2^k} W(x + y, x - y),$$

In general, for r that is divisible by 4 we have

$$x^{n-r} y^r = x^{n-r} (iy)^r, \text{and}$$

for r that is even but not divisible by 4, we get

$$x^{n-r} y^r = -x^{n-r} (iy)^r.$$

Therefore, the weight enumerator of $C^{(0)}$ is

$$W_{C^{(0)}}(x, y) = \frac{1}{2^k} \left\{ \frac{W(x + y, x - y) + W(x + iy, x - iy)}{2} \right\}, \text{ or}$$

$$W_{C^{(0)}}(x, y) = \frac{1}{2^{k+1}} \{W(x + y, x - y) + W(x + iy, x - iy)\} \quad (4.9)$$

Then using the MacWilliams identity again, we have the weight enumerator of $C^{(0)\perp}$

$$W_{C^{(0)\perp}}(x, y) = \frac{1}{2^{n-(k+1)}} \left(\frac{1}{2^{k+1}} W_{C^{(0)}}(x + y, x - y) \right) \quad (4.10)$$

From (4.9) and (4.10) we have:

$$W_{C^{(0)\perp}}(x,y) = \frac{1}{2^n}\Big\{W((x+y)+(x-y),(x+y)-(x-y)) \\ +W((x+y)+i(x-y),(x+y)-i(x-y)\Big\}$$

Thus we have, $W_{C^{(0)\perp}}(x,y)$ is equal to

$$\frac{1}{2^n}\left\{W(2x,2y)+W((1+i)x+(1-i)y,(1-i)x+(1+i)y)\right\}$$

$W(x,y)$ is a homogeneous polynomial of degree n so that

$$\frac{1}{2^n}W(2x,2y) = W(x,y), \text{ and}$$

$$\frac{W((1+i)x+(1-i)y,(1-i)x-(1+i)y)}{2^n} \\ = W\Big(\frac{(1+i)x+(1-i)y}{2},\frac{(1-i)x+(1+i)y}{2}\Big)$$

Thus the weight enumerator of the shadow code $S = C^{(0)\perp} \setminus C$ ($S(x,y)$ is equal to

$$\left\{W(x,y)+W\left(\frac{(1+i)x+(1-i)y}{2},\frac{(1-i)x+(1+i)y}{2}\right)\right\}-W(x,y).$$

Thus,

$$S(x,y) = W\left(\frac{(1+i)x+(1-i)y}{2},\frac{(1-i)x+(1+i)y}{2}\right).$$

Again, using MacWilliams identity and the property that $W(x,y)$ is homogeneous of degree n, we have

$$S(x,y) = \frac{2^k}{2^n}W^{\perp}\Big(\frac{[1+i]x+[1-i]y+[1-i]x+[1+i]y}{2}, \\ \frac{[1+i]x+[1-i]y-[1-i]x-[1+i]y}{2}\Big) \\ = \frac{1}{2^{n-k}}W^{\perp}(x+y,i(x-y)).$$

□

The following theorem will summarize the properties of the shadow code of a type I self-dual code derived from Theorem 4.1.

Theorem 4.2. *Let C be an [n, n/2, d] Type I self-dual code. The dual $C^{(0)\perp}$ consists of the union of four cosets of $C^{(0)}$, say $C^{(0)} \cup C^{(1)} \cup C^{(2)} \cup C^{(3)}$, with $C = C^{(0)} \cup C^{(2)}$.*

(i) S is a nonlinear code, given by

$$S = C^{(0)} \setminus C = C^{(1)} \cup C^{(3)}. \tag{4.11}$$

(ii) If $u,\ v \in S$ then $u + v \in C$. More precisely, if $u, v \in C^{(1)}$ then $u+v \in C^{(0)}$; if $u \in C^{(1)}, v \in C^{(3)}$ then $u+v \in C^{(2)}$; and if $u, v \in C^{(3)}$ then $u + v \in C^{(0)}$.

(iii) Let $S(x, y) = \sum B_r x^{n-r} y^r$ be the weight enumerator of S. Then B_r are nonnegative integers satisfying $B_r = B_{n-r}$ for all r, and

$$S(x, y) = W\left(\frac{x+y}{\sqrt{2}}, \frac{i(x-y)}{\sqrt{2}}\right) \tag{4.12}$$

where $W(x, y)$ is the weight enumerator of C.

$$B_r = 0, \ \textit{unless } r \equiv n/2 \ (\textit{mod } 4), \tag{4.13}$$

$$B_0 = 0, \tag{4.14}$$

$$B_r \leq 1 \qquad \textit{for } r < d/2, \tag{4.15}$$

$$B_{d/2} \leq 2n/d, \tag{4.16}$$

$$B_r \leq A(n,d,r), \qquad \textit{for all } r, \textit{ and} \tag{4.17}$$

$$\textit{at most one } B_r \textit{is nonzero for } r < (d+4)/2. \tag{4.18}$$

(iv) If $W(x,y)$ are the weight enumerators of C and if we write

$$W(x,y) = \sum_{j=0}^{\lfloor \frac{n}{8} \rfloor} a_j (x^2+y^2)^{n/2-4j} (x^2y^2(x^2-y^2)^2)^j \tag{4.19}$$

using Gleason's theorem, for suitable a_i, then

$$S(x,y) = \sum_{j=0}^{\lfloor \frac{n}{8} \rfloor} (-1)^j a_j (2)^{n/2-6j} (xy)^{n/2-4j} (x^4-y^4)^{2j} \tag{4.20}$$

In particular, a_j is divisible by $2^{n/2-6j}$ for all i.

Proof. Since C is an $[n, n/2, d]$ Type I self-dual code, then $C = C^{\perp}$ and $C^{(0)\perp} = C^{(0)} \cup C^{(1)} \cup C^{(2)} \cup C^{(3)}$, where $C = C^{(0)} \cup C^{(2)}$, Then from (4.1) we have (i), $S = C^{(1)} \cup C^{(3)}$. Consider the abelian group $C^{(0)\perp}/C^{(0)}$ of order 4. The group is not cyclic since it is the quotient of $C^{(0)\perp}$ that is a vector space over $\mathbb{F}_2$. Thus $C^{(0)\perp}/C^{(0)}$ should be the Klein-4 group so that (ii) follows. (4.12) follows from (4.8) since $W^{\perp} = W$, (4.14) - (4.18) follow from (4.2) - (4.5). (4.20) follows from (4.12) and (4.19) and (4.13) follows from (4.20). □

Using Theorem 4.2, we will now construct all the Type I extremal and optimal weight enumerator of length 38 and 40. We will use the properties (4.19), (4.20), in Theorem 4.2 to construct the Type I extremal weight enumerator of length 38 and 40 and

use the properties (4.14), (4.15), and (4.16) in the theorem to show that the extremal weight enumerators for both lengths do not exist. Then we construct the optimal weight enumerator for each length.

(i) The weight enumerator of Type I Self-dual codes of length 38. We have $\lfloor \frac{38}{8} \rfloor = 4$, and using Theorem 1.1, the extremal weight enumerator has minimum weight 10 (if exists). We construct the weight enumerator of the extremal code iteratively. By Gleason's theorem we have,

$$W(x,y) = \sum_{j=0}^{4} a_j (x^2+y^2)^{19-4j} (x^2y^2(x^2-y^2)^2)^j$$

Expanding the equation we have

$$W(x,y) = a_0 + (19a_0 + a_1)y^2 + (171a_0 + 13a_1 + a_2)y^4 \cdots$$

We will choose a_k such that we have the extremal weight enumerator $W(x,y) = 1 + Ay^{10} + \cdots$. We start by choosing $a_0 = 1$. This is the only possible choice since the code has only one codeword $\mathbf{0}$. Then we have

$$W(x,y) = 1 + (19 + a_1)y^2 + (171 + 13a_1 + a_2)y^4 \cdots$$

Substituting $a_1 = -19$, the coefficient of y^2 vanishes and we have

$$W(x,y) = 1 + (-76 + a_2)y^4 + (-475 + 7a_2 + a_3)y^6 + \cdots$$

Substituting $a_2 = 76$ to make the coefficient of y^4 equal to 0, we obtain

$$W(x,y) = 1 + (57 + a_3)y^6 + (228 + a_3 + a_4)y^8 + \cdots$$

Substituting $a_3 = -57$ to eliminate the coefficient of y^6, the weight enumerator that we have is

$$W(x,y) = 1+(171+a_4)y^8+(1862-5a_4)y^{10}+7(1482+a_4)y^{12}+\cdots$$

Assume that the extremal code exists, then we substitute $a_4 = -171$ so that the coefficient of y^8 is vanished and we get

$$\begin{aligned} W(x,y) &= 1 + 2717y^{10} + 9177y^{12} + 35910y^{14} \\ &+88521y^{16} + 125818y^{18} + 125818y^{20} + 88521y^{22} \\ &+35910y^{24} + 9177y^{26} + 2717y^{28} + y^{38} \end{aligned} \tag{4.21}$$

Now, we consider the weight enumerator of the shadow. Using the equation (4.20) in Theorem 4.2, we have

$$S(x,y) = \sum_{j=0}^{4} a_j(-1)^j 2^{(19-6j)} y^{19-4j}(1-y^4)^{2j}$$

Expanding the equation we have

$$S(x,y) = \frac{a_4}{32}y^3 + \left(-2a_3 - \frac{a_4}{4}\right)y^7 + 128a_2 + 12a_3 + 7\frac{a_4}{8}\Big)\, y^{11} + \cdots$$

From the weight enumerator of the shadow we see that all coefficients of that polynomial will be integers if and only if the value of a_4 is a multiple of 32. Substituting $a_0 = 1$, $a_1 = -19$, $a_2 = 76$,

and $a_3 = -57$ as what we did on the iterative construction of the weight enumerator $W(x, y)$, we get

$$\begin{aligned} S(x,y) = \frac{a_4}{32}y^3 + \left(114 - \frac{a_4}{4}\right)y^7 + \left(9044 + 7\frac{a_4}{8}\right)y^{11} \\ + \left(118446 - 7\frac{a_4}{4}\right)y^{15} + 35\left(7688 + \frac{a_4}{16}\right)y^{19} \\ + \left(118446 - 7\frac{a_4}{4}\right)y^{23} + \left(9044 + 7\frac{a_4}{8}\right)y^{27} \\ + \left(114 - \frac{a_4}{4}\right)y^{31} + \frac{a_4}{32}y^{35} \end{aligned}$$

For the coefficients of $S(x, y)$ to be integers, the possible values of a_4 are $a_4 = 32m$, for some $m \geq 0$. If $a_4 = 0$, then

$$\begin{aligned} S(x,y) = 114y^7 + 9044y^{11} + 118446y^{15} + 269080y^{19} \\ +118446y^{23} + 9044y^{27} + 114y^{31}, \end{aligned} \quad (4.22)$$

and if $a_4 = 32m$, for every positive integer m we get

$$\begin{aligned} S(x,y) = my^3 + (114 - 8m)y^7 + (9044 + 28m)y^{11} \\ +(118446 - 56m)y^{15} \cdots . \end{aligned}$$

Since for $a_4 = -171$ the weight enumerator of the shadow has noninteger coefficients, we can conclude that the extremal Type I code of length 38 with weight enumerator (4.21) does not exist. Therefore the optimal self-dual code of this length is the code with minimum weight 8. By property (4.15) in the Theorem 4.2 we should have $m = 1$, and the weight enumerator of the shadow is

$$\begin{aligned} S(x,y) = y^3 + 106y^7 + 9072y^{11} + 118390y^{15} + 269150y^{19} \\ +118390y^{23} + 9072y^{27} + 106y^{31} + y^{35}. \end{aligned}$$

Thus there are 2 possible weight enumerator for self-dual code of lenght 38 with optimal minimum weight 8. If $a_4 = 0$, then

$$\begin{aligned} W(x,y) = 1 &+ 171y^8 + 1862y^{10} + 10374y^{12} + 36765y^{14} \\ &+84759y^{16} + 128212y^{18} + 128212y^{20} + 84759y^{22} \\ &+36765y^{24} + 10374y^{26} + 1862y^{28} + 171y^{30} + y^{38}. \end{aligned}$$

If $a_4 = 32$, then

$$\begin{aligned} W(x,y) = 1 &+ 203y^8 + 1702y^{10} + 10598y^{12} + 36925y^{14} \\ &+84055y^{16} + 128660y^{18} + 128660y^{20} + 84055y^{22} \\ &+36925y^{24} + 10598y^{26} + 1702y^{28} + 203y^{30} + y^{38} \end{aligned}$$

(ii) The weight enumerator of Type I Self-dual code of length 40. We have $\lfloor \frac{40}{8} \rfloor = 5$, and using Theorem 1.1, the extremal weight enumerator has minimum weight 12 (if exists). Again using Gleason's theorem we have,

$$W(x,y) = \sum_{j=0}^{5} a_j (x^2+y^2)^{19-4j} (x^2y^2(x^2-y^2)^2)^j$$

Expanding the equation we have

$$W(x,y) = a_0 + (20a_0 + a_1)y^2 + (1140a_0 + 89a_1 + 8a_2 + a_3)y^6 + \cdots$$

We start choosing a_j such that we have the extremal weight enumerator $W(x,y) = 1 + Ay^{12} + \cdots$. There is only one choice for a_0, since there is only one vector $\mathbf{0}$ in the code. Substituting $a_0 = 1$ we have

$$W(x,y) = 1 + (20 + a_1)y^2 + (190 + 14a_1 + a_2)y^4 + \cdots$$

Substituting $a_1 = -20$ to make the coefficient of y^2 equal to 0, we have

$$W(x,y) = 1 + (-90 + a_2)y^4 + (-640 + 8a_2 + a_3)y^6 + \cdots$$

Substituting $a_2 = 90$ to eliminate the coefficient of y^4, we obtain

$$W(x,y) = 1 + (80 + a_3)y^6 + (285 + 2a_3 + a_4)y^8 + \cdots$$

Substituting $a_3 = -80$ to reduce the coefficient of y^6 to be 0, we have

$$W(x,y) = 1 + y^8(125 + a_4) + y^{10}(1664 - 4a_4 + a_5) + \cdots$$

Then we substitute $a_4 = -125$ so that the coefficient of y^8 is vanished, we get

$$W(x,y) = 1 + (2164 + a_5)y^{10} + 2(5235 - 5a_5)y^{12} + 3(14220 + 15a_5)y^{14} + \cdots \quad (4.23)$$

Assumed the extremal code exists, then substituting $a_5 = -2164$, we get

$$\begin{aligned} W(x,y) = 1 &+ 32110y^{12} - 54720y^{14} + 381615y^{16} \\ -237120y^{18} &+ 804804y^{20} - 237120y^{22} + 381615y^{24} \\ &-54720y^{26} + 32110y^{28} + y^{40} \end{aligned} \quad (4.24)$$

Now, we consider the weight enumerator of the shadow. Using the equation (4.20) in Theorem 4.2, we have

$$S(x,y) = \sum_{j=0}^{5} a_j(-1)^j 2^{(20-6j)} y^{20-4j}(1 - y^4)^{2j}$$

Expanding the equation we have

$$S(x,y) = -\frac{a_5}{1024} + \frac{32a_4 + 5a_5}{512}y^4 + \left(-4a_3 - \frac{a_4}{2} - \frac{45a_5}{1024}\right)y^8 + \cdots$$

By property (4.14) of Theorem 4.2 that require $B_0 = 0$ we should have $a_5 = 0$ to get

$$\begin{aligned} S(x,y) = \frac{a_4}{16}y^4 + \left(-4a_3 - \frac{a_4}{2}\right)y^8 + \left(256a_2 + 24a_3 + 7\frac{a_4}{4}\right)y^{12} \\ + \left(-16384a_1 - 1024a_2 - 60a_3 - 7\frac{a_4}{2}\right)y^{16} \\ + \left(1048576a_0 + 32768a_1 + 1536a_2 + 80a_3 + 35\frac{a_4}{8}\right)y^{20} + \cdots \end{aligned}$$

This means that the extremal weight enumerator that have been constructed in (4.24) for $a_5 = -2164$, is impossible. In other words, we can not vanish the coefficient of y^{10} in the weight enumerator. Therefore at this stage the optimal weight enumerator that is possible is the one with minimum weight 10. For the coefficients of $S(x,y)$ to be integers, we must have $a_4 = 16m$, for $m \geq 0$. This implies that our previous choice $a_4 = -125$ to eliminate the coefficient of y^8 in the weight enumerator $W(x,y)$ is also impossible. So the optimal code is the one that has minimum weight 8.

For $a_4 = 16m$, we get

$$\begin{aligned} S(x,y) = my^4 + (228 - 8m)y^8 + 28(646 + m)y^{12} \\ +(236892 - 56m)y^{16} + 70(7688 + m)y^{20} + \cdots . \end{aligned}$$

By property (4.15) of Theorem 4.2, the coefficient of $y^{d/2} = y^4$ in the weight enumerator of the shadow should be less than or equal

$2n/d = 10$, therefore $m \leq 10$. The possible weight enumerators could be constructed by substituting the possible values of a_4 i.e., $a_4 = 16m$, for $0 \leq m \leq 10$. Thus we have the possible weight enumerators of the self-dual code of length 40 with optimal minimum weight 8 are

$$\begin{aligned} W(x,y) = 1 &+ (125+16m)y^8 - 64(-26+m)y^{10} \\ &+32(335+m)y^{12} + 192(230+m)y^{14} + (119810-272m)y^{16} \\ &-128(-1690+m)y^{18} + 448(587+m)y^{20} - 128(-1690+m)y^{22} \\ &+(119810-272m)y^{24} + 192(230+m)y^{26} + 32(335+m)y^{28} \\ &-64(-26+m)y^{30} + (125+16m)y^{32} + y^{40} \end{aligned}$$

where $0 \leq m \leq 10$.

The weight enumerators resulted from our calculation are the known weight enumerators of self-dual binary codes of length 38 and 40, see [8].

5. GABORIT AND OTMANI EXPERIMENTAL CONSTRUCTION

As mentioned in section 1, in this section we will discuss a relatively new experimental method to construct self-dual binary codes of any length which was proposed by P. Gaborit and A. Otmani [6] as a generalization of the binary construction that was done by J.C.Carlach, A. Otmani, and C. Vervoux [3], [5].

As we mentioned in section 1, two binary codes are equivalent if they differ only in the order of coordinates of the codewords. For any permutation σ of S_n, and for C^σ be the code

$$\{(c_{\sigma(1)}, c_{\sigma(2)}, \cdots, c_{\sigma(n)} | (c_1, \cdots, c_n) \in C\}$$

then C and C^σ are equivalent. The set of all permutations that preserve the code C forms a group that is called the automorphism group $Aut(C)$, which is a subgroup of S_n. In our experiment, we found that the Gaborit and Otmani construction sometimes gives some new codes that are equivalent. Using GAP programming that is modified from the program written by M. Armentrout, J. Thomas et. al. [18], we execute the construction of self-dual codes of several lengths and get some codes with min-

imum weights close to the extremal weights of those lengths.

Now we will discuss the scheme of the construction that is given in [5] and [3] for general linear binary codes that have information rate 1/2. This construction has been generalized as Gaborit and Otmani's experimental construction. Let C_b be a code of dimension k_b of length $2k_b$ and suppose that a generator matrix of C_b is $[I_{k_b}|P_b]$. Let k be a multiple of k_b (say, $k = ek_b$) and consider a finite sequence of permutation $\Pi = (\pi_1, \cdots, \pi_s)$ of the symmetric group S_k where s is a nonnegative integer. We would like to compute k redundant bits from a message of k useful information bits $X = (x_0, x_1, \cdots, x_{k-1})$ to form a 1/2-rate code C. Then we split X into e messages $X = (X_1, \cdots, X_e)$ of length k_b so that each of them is encoded through C_b to form a vector Y_j of length k_b, where $j = 1, \cdots, e$. In other words $Y_j = X_j C_b$, for every $j = 1, \cdots, e$. Thus if P are the block matrices

$$P = \begin{pmatrix} P_b & 0 & \dots & 0 \\ 0 & P_b & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & P_b \end{pmatrix}, \tag{5.1}$$

where each 0 is a zero $k_b \times k_b$ square matrix and $Y = (Y_1, \cdots, Y_e)$, then $Y = XP$. Then we multiply the vector Y to a permutation matrix Π_1 to get a new vector $Y\Pi_1$. Next we do the same encoding

process for $Y\Pi_1$ as what we do for X to get another vector

$$R^{(1)}(X) = Y\Pi_1 P = XP\Pi_1 P.$$

We repeat this process as many times as the number of permutations that we have (s) so that at the end of the process we have the computed vector

$$R^{(s)}(X) = XP\Pi_1 P\Pi_2 P \cdots \Pi_s P.$$

Consider the set of vectors of length $2k$ defined by

$$C = \{XR^{(s)}(X); X \in \mathbb{F}_2^k\}.$$

where $XR^{(s)}(X)$ is a concatenation of the vectors X and $R^{(s)}(X)$. Then C is a code of length $2k$ of dimension k, with generator matrix

$$G = (I_k | P\Pi_1 P\Pi_2 P \cdots \Pi_s P).$$

Now we consider the construction when C_r is a self dual code.

Theorem 5.1. *If C_r is self-dual then for any finite sequence of permutations $\pi = (\pi_1, \cdots, \pi_s)$ of S_k the code*

$$C = \{XR^{(s)}(X); X \in \mathbb{F}_2^k\}.$$

is self-dual.

Proof. We use the fact that C is self-dual if and only if it has a generator matrix of the form $G = [I_k|B]$ such that

$$B^T B = BB^T = I_k \tag{5.2}$$

where B^T is the transpose matrix of B. Since C_b is self-dual then it has generator matrix $[I_k|P_b]$, such that

$$P_b^T P_b = I_{k_r}.$$

Thus,

$$P^T P = I_k,$$

where P has the same form with (5.1). For any permutation matrix Π_i of size k, we also have

$$\Pi_i^T \Pi_i = \Pi_i \Pi_i^T = I_k,$$

so that we have C with generator matrix

$$G = (I_k|P\Pi_1 P\Pi_2 P \cdots \Pi_s P)$$

is also self-dual. □

The next theorem gives properties of the self-dual code constructed by this method.

Theorem 5.2. *Let $\pi = (\pi_1, \cdots, \pi_s)$ be a finite sequence of permutations in S_k and C_b be a Type II code with generator matrix $[I_k|P_b]$. If the number of permutations s is even then the code C that has generator matrix $G = (I_k|P\Pi_1 P\Pi_2 P \cdots \Pi_s P)$ is a Type II code, where P is of the form in* (5.1) *. If the number of permutations s is odd then the code C that has generator matrix $G = (I_k|P\Pi_1 P\Pi_2 P \cdots \Pi_s P)$ is a Type I code.*

Proof. If v_1 and v_2 are the vectors of $\mathbb{F}_2^{k_r}$ such that $v_2 = P_r v_1$, then

$$wt(v_1) + wt(v_2) \equiv 0 \ (mod\ 4). \tag{5.3}$$

Property (5.3) is still true for any vectors z_1 and z_2 of $\mathbb{F}_2^k$ such that $z_2 = Pz_1$ if P is defined as in the relation (5.1). So any codeword $XR^{(s)}(X)$ of C satisfies the following relation:

$$wt(X) \equiv (-1)^{s+1} wt(R^{(s)}(X)) \ (mod\ 4) \tag{5.4}$$

Thus, if s is even, the weight of $XR^{(s)}(X)$ is a multiple of 4, if not $wt(XR^{(s)}(X))$ is congruent to 2 modulo 4 whenever $wt(X)$ is odd.

□

Generalizing the above construction, F. Gaborit and A. Otmani in [6] start with a self-dual code C of length n over a ring $\mathbf{R}$, to construct a self-dual code of the same length with greater minimum distance. If in the method given by J.C.Carlach, A.

Otmani, and C. Vervoux we construct a self-dual code from the certain simple self-dual code of smaller length which has generator matrix $G = [I_k|P_b]$, in Gaborit and Otmani experimental method we start with a simple known self-dual code of length n. The original self-dual code is built by taking the direct sum of the simple code $i_2 = \{(0,0),(1,1)\}$. Thus, in this method, to construct a self-dual code of length n, we start with the self-dual code C with the $n \times n$ generator matrix

$$G = \begin{pmatrix} 1 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & 1 \end{pmatrix}.$$

To be consistent, we will concentrate on the case for self-dual binary codes. Let G be the generator matrix of C, and let M_i be the $n \times n$ orthogonal matrices, i.e. the matrices that satisfy

$$M_i \cdot M_i^T = I_n.$$

If $\Pi_1, \cdots \Pi_r$ are permutations of the symmetric group S_n where $r \geq 1$, then we get a code C_r with generator matrix

$$G_r = GM_1\Pi_1 \cdots M_r\Pi_r.$$

Gaborit and Otmani stated that a sequence of permutations $\pi_1, \ldots, \pi_r$ belongs to one of the three families f_j $(1 \leq j \leq 3)$ if there exists an invertible element a in $\mathbb{Z}_n$ such that for any $1 \leq i \leq r$, π_i

satisfies the relations:

$$f_1 : \mathbb{Z}_n \to \mathbb{Z}_n, \text{ such that } \pi_i(z) = a \cdot (z+1),$$

$$f_2 : \mathbb{Z}_n \to \mathbb{Z}_n, \text{ such that } \pi_i(z) = a^i \cdot (z+1),$$

and

$$f_3 : \mathbb{Z}_n \to \mathbb{Z}_n, \text{ such that } \pi_i(z) = a^i \cdot (z+i).$$

Any sequence of permutations that belongs to the family f_i is completely determined by a and r. Therefore a code constructed from a sequence belonging to f_i is said as obtained from the $(f_i; a; r)$ construction.

Theorem 5.3. *If C is self-dual then so is C_r.*

If $M_i = M$ for every $i = 1, \cdots, r$ then this theorem is similar with Theorem 5.1. In general the theorems follow from the following lemma.

Lemma 5.1. *Let G be a generator matrix of a self-dual code of length n over a ring $\mathbf{R}$ and let M be a $n \times n$ matrix over $\mathbf{R}$ satisfying $M \cdot M^T = \lambda I_n$, with λ an invertible element of $\mathbf{R}$. Then the code with generator matrix GM is self-dual.*

Proof. Since M is invertible then the number of codes generated by GM is the same with the number of codes generated by G. The code with generator matrix G is self-dual so that $G \cdot G^T = 0$. Thus $(GM) \cdot (GM)^T = G \cdot (MM^T)G^T = \lambda GG^T = 0$ which proves the code with generator matrix GM is self-dual. □

For constructing self-dual codes of length ≤ 130, Gaborit and Otmani build the orthogonal matrix M using the 4×4 orthogonal matrix

$$B = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}.$$

Then we have the orthogonal matrix

$$M = \begin{pmatrix} B & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & B \end{pmatrix},$$

if $n = 0(mod\ 4)$, or

$$M = \begin{pmatrix} B & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \cdots & B & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & I_2 \end{pmatrix},$$

if $n = 2(mod\ 4)$.

Except for $n = 54,\ 66,\ 78,\ 102,\ 114$, and 126, they use the 6×6

matrix

$$B_6 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

Then we get

$$M = \begin{pmatrix} B_6 & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \cdots & B_6 & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & I_{k_2} \end{pmatrix},$$

where $k_2 = n - 6\lfloor n/6 \rfloor$. They did not give the reason behind their choices.

This construction goal is to get a code with high minimum weight and the procedure is quite simple and easy to understand, especially in engineering point of view. Mathematically, we consider the mathematical arguments that are used in taking the permutations, in choosing the orthogonal matrices and also in taking the initial simple generator matrix to construct the new code. How do all of those matrices involve in the construction and which one gives the significant contribution to build the good code. From our experiments we do not get mathematical arguments to answer

those questions. One fact that is easy to explained in taking the orthogonal matrix is the matrix must not be a permutation matrix. Since, if we take the permutation matrix M, then applying the Gaborit and Otmani experimental method, the result code C_r will be equivalent with the original code C. We will not get a better code with higher minimum weight. So far, there is no articles that discuss about the mathematical analysis of the Gaborit and Otmani experimental construction that could be refered.

In our experiment, we tried to find out how orthogonal matrix chosen in the Gaborit and Otmani experimental construction affect the result. We applied the construction with both orthogonal matrices given in [6] to construct self-dual binary codes of lengths up to 66. The result of our experiments are listed in Table (5.1) and Table (5.2). We give an example of our experiment for $n = 12$ with both orthogonal matrix B and B_6 in Appendix I.

From the result, we get that the choice of orthogonal matrices do not give significant effects. For almost all lengths we can get a code with a high minimum weight using both orthogonal matrices. Although in our result the experiment with matrix B gives the better minimum weights for some lengths, it simply because we do not enough time to try all possible (f_i, a, r) constructions for the experiment with matrix B_6 for codes of large lengths. For example, to execute one (f_i, a, r) construction for a code with

Tab. 5.1: Self-dual binary codes constructed by different orthogonal matrices

n	Construction$(f_i; a; r)$;B	d	Construction$(f_i; a; r)$;B_6	d
2	(f_1;1;1)	2	(f_1;1;1)	2
4	(f_1;1;1)	2	(f_1;1;1)	2
6	(f_1;1;1)	2	(f_1;1;1)	2
8	(f_1;1;1)	2	(f_1;1;1)	2
10	(f_1;1;1)	2	(f_1;1;1)	2
12	(f_1;1;4)	4	(f_1;5;2)	4
14	(f_1;1;4)	4	(f_1;3;3)	4
16	(f_1;1;4)	4	(f_1;3;3)	4
18	(f_1;1;4)	4	(f_1;5;2)	4
20	(f_1;1;4)	4	(f_1;3;3)	4
22	(f_1;1;20)	6	(f_1;3;3)	4
24	(f_1;1;4)	6	(f_1;5;3)	6
26	(f_1;3;3)	6	(f_1;5;15)	6
28	(f_1;1;4)	6	(f_1;5;13)	6
30	(f_1;1;17)	6	(f_1;7;3)	6
32	(f_1;3;3)	8	(f_1;5;5)	6
34	(f_1;3;4)	6	(f_1;3;3)	6
36	(f_1;5;5)	8	(f_3;5;3)	8
38	(f_1;1;116)	8	(f_3;3;4)	6
40	(f_1;3;3)	8	(f_1;3;4)	6
42	(f_1;1;94)	8	(f_1;5;3)	8
44	(f_1;3;3)	8	(f_1;3;5)	8
46	(f_1;3;6)	8	(f_1;3;6)	8
48	(f_1;5;3)	8	(f_1;5;7)	8
50	(f_1;3;3)	8	(f_1;3;5)	8
52	(f_1;1;19)	10	(f_1;3;4)	8
54	(f_1;5;5)	8	(f_1;5;4)	8
56	(f_1;1;10)	10	(f_1;3;4)	8

length 64 we spent 20 minutes to get the result and for a code with length 66 we need 48 minutes. Therefore to execute all possible (f_i, a, r) constructions to get the best code of larger length we will need a lot of time. For the experiment with matrix B, we have already had the Gaborit and Otmani's data which showed all the (f_i, a, r) constructions which give the best result. Applying the data, we can get the best code faster. Probably if we try all possible (f_i, a, r) constructions using matrix B_6, we will also get the best codes of every length. On the other hand it is also possible that this construction with orthogonal matrix B_6 misses the best codes of some lengths. There is no mathematical theories to explain the probability of getting the best code with this method related to the choice of orthogonal matrices.

Tab. 5.2: Self-dual binary codes constructed by different orthogonal matrices (continued)

n	Construction$(f_i; a; r)$;B	d	Construction$(f_i; a; r)$;B_6	d
58	(f_1;1;112)	10	(f_1;3;4)	8
60	(f_1;1;78)	12	(f_1;7;3)	10
62	(f_1;1;21)	10	(f_1;3;5)	8
64	(f_1;3;11)	10	(f_1;3;11)	10
66	(f_1;7;23)	10	(f_1;7;22)	10

We also try to see how the choices of a and r affect the construction of self-dual code of length 32 using orthogonal matrix B. Data for this experiments can be seen in Table (5.3). We found

that for $a = 3, 11,$ and 19 we get the codes with the same minimum weights for every r. For those values of a and $r = 3$ we also have the code with largest minimum weight for this length (8). We also checked that the codes obtained for this choice of a and r are equivalent (see Appendix II). But we can not give the theoretical explanation for this result. There is no pattern that can be generalized.

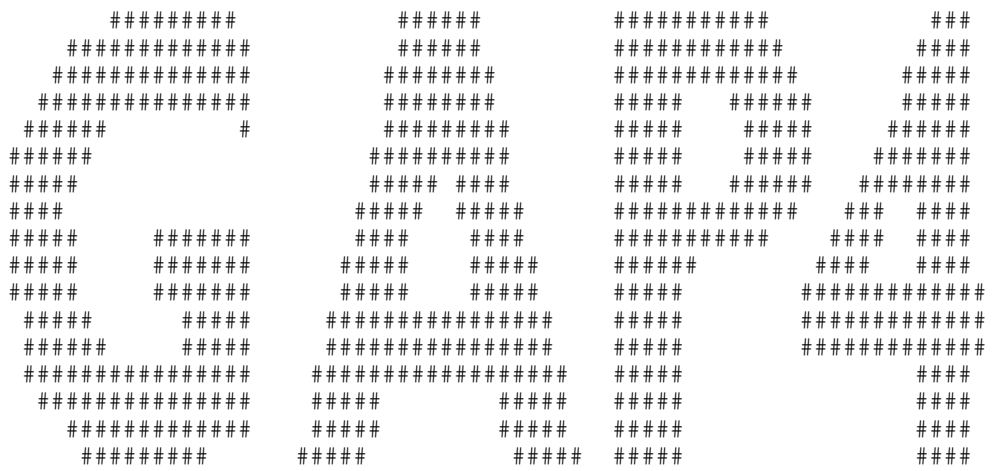
Tab. 5.3: $(f_1; a; r)$ Construction for $n = 32$

a	r	d	a	r	d	a	r	d	a	r	d
1	1	2	5	1	2	9	1	2	15	1	2
1	2	2	5	2	6	9	2	6	15	2	2
1	3	2	5	3	6	9	3	6	15	3	2
1	4	6	5	4	4	9	4	6	15	4	2
1	5	6	5	5	4	9	5	6	15	5	2
1	6	4	5	6	4	9	6	6	15	6	2
1	7	4	5	7	4	9	7	6	15	7	2
1	8	4	5	8	4	9	8	4	15	8	2
1	9	4	5	9	4	9	9	4	15	9	2
1	10	4	5	10	6	9	10	6	15	10	2
3	1	2	7	1	2	11	1	2	17	1	2
3	2	4	7	2	4	11	2	4	17	2	4
3	3	8	7	3	4	11	3	8	17	3	4
3	4	4	7	4	2	11	4	4	17	4	6
3	5	4	7	5	2	11	5	4	17	5	6
3	6	8	7	6	4	11	6	8	17	6	4
3	7	4	7	7	4	11	7	4	17	7	4
3	8	2	7	8	2	11	8	2	17	8	4
3	9	2	7	9	2	11	9	2	17	9	4
3	10	4	7	10	4	11	10	4	17	10	4
19	1	2	19	2	4	19	3	8	19	4	4
19	5	4	19	6	8	19	7	4	19	8	2
19	9	2	19	10	4						

Appendix I

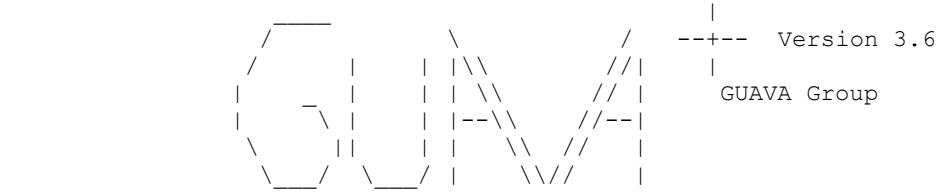
Example Gaborit-Otmani Experimental Construction $((f_1, 1, r)$ with matrix B and $(f_1, 5, r)$ with matrix $B_6)$ for $n = 12$.

```
lab47:~~$ gap
```



```
     Information at:  http://www.gap-system.org
     Try '?help' for help. See also  '?copyright' and  '?authors'

   Loading the library. Please be patient, this may take a while.
GAP4, Version: 4.4.10 of 02-Oct-2007, i486-pc-linux-gnu-i486-linux-gnu-gcc
Components:  small 2.1, small2 2.0, small3 2.0, small4 1.0, small5 1.0,
             small6 1.0, small7 1.0, small8 1.0, small9 1.0, small10 0.2,
             id2 3.0, id3 2.1, id4 1.0, id5 1.0, id6 1.0, id9 1.0, id10 0.1,
             trans 1.0, prim 2.1  loaded.
Packages:    CTblLib 1.1.3, TomLib 1.1.2  loaded.
gap> LoadPackage("guava");
```



```
true
gap> # This program executes the algorithm designed
gap> # by Philippe Gaborit and Ayoub Otmani to construct self-dual codes.
```

```
gap> # The algorithm also computes the greatest minimum distance
gap> # between codes to find the strongest self-dual codes.
gap>
gap> # This function computes the dot product of two vectors, vecA and vecB.
gap> DotProd := function(vecA, vecB)
>       local ii, tot;
>
>       tot := vecA[1] * vecB[1];
>       for ii in [2..Length(vecA)] do
>               tot := tot + vecA[ii] * vecB[ii];
>       od;
>
>       return tot;
> end;
function( vecA, vecB ) ... end
gap>
gap> # This function constructs the nxn matrix, M, given a matrix B such
gap> # that B*B^T = c * I, the length of the code, n, and the field in
gap> # The matrix M is constructed by placing copies of B along
gap> # the diagonal until B no longer fits at which point the diagonal
gap> # entries are comprised of a scalar matrix b*I, where b^2 = c.
gap>
gap> Build_M_Mat := function(B, n, field)
>       local b, beta, lambda, basemat,
>               blocks, row, ii, M;
>
>       b := Length(B);
>
>       # Special case (just a scalar mat)...
>       # This creates the matrix M in the case where b >n.
>       if Int(n/b) = 0 then
>               lambda := DotProd(B[1], B[1]);
>               beta := First(Elements(field), x -> x*x = lambda);
>               M := beta * IdentityMat(n, field);
>               return M;
>       fi;
>
>       # Normal Case...
>       # This creates the matrix M in the case where b =< n.
>       basemat := IdentityMat(Int(n/b), field);
>       M := KroneckerProduct(basemat,B);
>
>       if n mod b > 0 then
>               lambda := DotProd(B[1], B[1]);
```

```
>               beta := First(Elements(field), x -> x*x = lambda);
>
>               for row in M do
>                       Append(row, List([1..(n mod b)], x -> Zero(field)));
>               od;
>
>               for ii in [1..(n mod b)] do
>                       row := List([1..n], x -> Zero(field));
>                       row[Int(n/b)*b+ii] := beta;
>                       Add(M, row);
>               od;
>       fi;
>
>       return M;
> end;
function( B, n, field ) ... end
gap>
gap> # Build up the permutation matrix Pi, representing
gap> # how the function pi permutes the elements of Z_n
gap> Build_Pi_Mat := function(pi, n, field)
>       local perm, images;
>
>       images := List([0..(n-1)], x -> pi(x));
>       images := List(images, x -> x + 1);
>
>       perm := PermList(images);
>       return PermutationMat(perm, n, field);
> end;
function( pi, n, field ) ... end
gap>
gap> # This function executes the algorithm for creating
gap> # a new self-dual generating matrix for another self-dual code.
gap> # Given an initial self-dual generating matrix G, the previously
gap> # mentioned matrix B, three parameters, f, a, and r, and the field.
gap> MakeCodeMat := function(G, B, f, a, r, field)
>       local M, n, pi, Pi, PiFuncs, Mats;
>
>       if not f in [1..3] then
>               Print("Invalid input for f!\n");
>               return fail;
>       fi;
>
>       n := Length(G[1]);
>       M := Build_M_Mat(B, n, field);
```

```
>
>       if not GCD_INT(a,n) = 1 then
>               Print("Error: n and a must be coprime!");
>               return fail;
>       fi;
>
>       if f = 1 then
>               pi := x -> a*(x+1) mod n;
>               Pi := Build_Pi_Mat(pi, n, field);
>
>               return  G * ((M * Pi) ^ r);
>       fi;
>
>       if f = 2 then
>               PiFuncs := List([1..r], i -> (x -> (a^i)*(x+1) mod n));
>       fi;
>
>       if f = 3 then
>               PiFuncs := List([1..r], i -> (x -> (a^i)*(x+i) mod n));
>       fi;
>
>       Mats := List(PiFuncs, p -> M * Build_Pi_Mat(p, n, field));
>       return G * Product(Mats);
> end;
function( G, B, f, a, r, field ) ... end
gap>
gap> B:= Z(2) * [[0,1,1,1],[1,0,1,1],[1,1,0,1],[1,1,1,0]];
[ [ 0*Z(2), Z(2)^0, Z(2)^0, Z(2)^0 ], [ Z(2)^0, 0*Z(2), Z(2)^0, Z(2)^0 ], [ Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0 ],
  [ Z(2)^0, Z(2)^0, Z(2)^0, 0*Z(2) ] ]
gap> G := Build_M_Mat([Z(2)*[1,1]],6,GF(2));
[ <a GF2 vector of length 12>, <a GF2 vector of length 12>, <a GF2 vector of length 12>, <a GF2 vector of length 12>,
  <a GF2 vector of length 12>, <a GF2 vector of length 12> ]
gap> G_C := MakeCodeMat(G,B,1,1,1,GF(2));
[ <a GF2 vector of length 12>, <a GF2 vector of length 12>, <a GF2 vector of length 12>, <a GF2 vector of length 12>,
  <a GF2 vector of length 12>, <a GF2 vector of length 12> ]
gap>  Display(G_C);
 . 1 1 . . . . . . . . .
 . . . 1 1 . . . . . . .
 . . . . . 1 1 . . . . .
 . . . . . . . 1 1 . . .
 . . . . . . . . . 1 1 .
 1 . . . . . . . . . . 1
gap> Display(G);
```

```
1 1 . . . . . . . . . . .
 . . 1 1 . . . . . . . . .
 . . . . 1 1 . . . . . . .
 . . . . . . 1 1 . . . . .
 . . . . . . . . 1 1 . . .
 . . . . . . . . . . 1 1 .
gap> Display(B);
 . 1 1 1
 1 . 1 1
 1 1 . 1
 1 1 1 .
gap> C := GeneratorMatCode(G_C,GF(2));
a linear [12,6,1..2]3..6 code defined by generator matrix over GF(2)
gap> MinimumWeight(C);
[12,6] linear code over GF(2) - minimum weight evaluation
Known lower-bound: 1
There are 2 generator matrices, ranks : 6 6
The weight of the minimum weight codeword satisfies 0 mod 4 congruence
Enumerating codewords with information weight 1 (w=1)
    Found new minimum weight 2
Number of matrices required for codeword enumeration 2
Completed w= 1, 12 codewords enumerated, lower-bound 4, upper-bound 2
Minimum weight: 2
2
gap> CodeWeightEnumerator(C);
x_1^12+6*x_1^10+15*x_1^8+20*x_1^6+15*x_1^4+6*x_1^2+1
gap> G_C := MakeCodeMat(G,B,1,1,2,GF(2));
[ <a GF2 vector of length 12>, <a GF2 vector of length 12>, <a GF2 vector of length 12>, <a GF2 vector of length 12>,
  <a GF2 vector of length 12>, <a GF2 vector of length 12> ]
gap> G_C := MakeCodeMat(G,B,1,1,1,GF(2));
[ <a GF2 vector of length 12>, <a GF2 vector of length 12>, <a GF2 vector of length 12>, <a GF2 vector of length 12>,
  <a GF2 vector of length 12>, <a GF2 vector of length 12> ]
gap> G_C2 := MakeCodeMat(G,B,1,1,2,GF(2));
[ <a GF2 vector of length 12>, <a GF2 vector of length 12>, <a GF2 vector of length 12>, <a GF2 vector of length 12>,
  <a GF2 vector of length 12>, <a GF2 vector of length 12> ]
gap> C2 := GeneratorMatCode(G_C2,GF(2));
a linear [12,6,1..2]3..6 code defined by generator matrix over GF(2)
gap>  Display(G_C2);
 . . 1 1 . . . . . . . .
 . 1 1 1 . . 1 1 1 . . .
 . . . . . . 1 1 . . . .
 1 . . . . 1 1 1 . . 1 1
 . . . . . . . . . . 1 1
 . . 1 1 1 . . . . 1 1 1
```

```
gap> MinimumWeight(C2);
[12,6] linear code over GF(2) - minimum weight evaluation
Known lower-bound: 1
There are 2 generator matrices, ranks : 6 6
The weight of the minimum weight codeword satisfies 0 mod 4 congruence
Enumerating codewords with information weight 1 (w=1)
    Found new minimum weight 2
Number of matrices required for codeword enumeration 2
Completed w= 1, 12 codewords enumerated, lower-bound 4, upper-bound 2
Minimum weight: 2
2
gap> CodeWeightEnumerator(C2);
x_1^12+6*x_1^10+15*x_1^8+20*x_1^6+15*x_1^4+6*x_1^2+1
gap> G_C3 := MakeCodeMat(G,B,1,1,3,GF(2));
[ <a GF2 vector of length 12>, <a GF2 vector of length 12>, <a GF2 vector of length 12>, <a GF2 vector of length 12>,
  <a GF2 vector of length 12>, <a GF2 vector of length 12> ]
gap> C3 := GeneratorMatCode(G_C3,GF(2));
a linear [12,6,1..2]3..6 code defined by generator matrix over GF(2)
gap> MinimumWeight(C3);
[12,6] linear code over GF(2) - minimum weight evaluation
Known lower-bound: 1
There are 2 generator matrices, ranks : 6 6
The weight of the minimum weight codeword satisfies 0 mod 4 congruence
Enumerating codewords with information weight 1 (w=1)
    Found new minimum weight 2
Number of matrices required for codeword enumeration 2
Completed w= 1, 12 codewords enumerated, lower-bound 4, upper-bound 2
Minimum weight: 2
2
gap> CodeWeightEnumerator(C3);
x_1^12+6*x_1^10+15*x_1^8+20*x_1^6+15*x_1^4+6*x_1^2+1
gap>  Display(G_C3);
 . . . 1 1 . . . . . . .
 1 1 . . . . . 1 1 . 1 1
 . . . . . . . 1 1 . . .
 1 . 1 1 1 1 . . . . . 1
 1 . . . . . . . . . . 1
 . . . 1 1 . 1 1 1 1 . .
gap> G_C4 := MakeCodeMat(G,B,1,1,4,GF(2));
[ <a GF2 vector of length 12>, <a GF2 vector of length 12>, <a GF2 vector of length 12>, <a GF2 vector of length 12>,
  <a GF2 vector of length 12>, <a GF2 vector of length 12> ]
gap> C4 := GeneratorMatCode(G_C4,GF(2));
a linear [12,6,1..4]3..4 code defined by generator matrix over GF(2)
gap>  Display(G_C4);
```

```
 . 1 1 1 . . 1 1 1 . . .
 . 1 1 . . 1 1 1 . . 1 .
 1 . . . . 1 1 1 . . 1 1
 . . 1 . . 1 1 . . 1 1 1
 . . 1 1 1 . . . . 1 1 1
 . 1 1 1 . . 1 . . 1 1 .
gap> MinimumWeight(C4);
[12,6] linear code over GF(2) - minimum weight evaluation
Known lower-bound: 1
There are 2 generator matrices, ranks : 6 6
The weight of the minimum weight codeword satisfies 0 mod 4 congruence
Enumerating codewords with information weight 1 (w=1)
    Found new minimum weight 4
Number of matrices required for codeword enumeration 2
Completed w= 1, 12 codewords enumerated, lower-bound 4, upper-bound 4
Minimum weight: 4
4
gap> CodeWeightEnumerator(C4);
x_1^12+15*x_1^8+32*x_1^6+15*x_1^4+1
gap> B_6:= Z(2) * [[1,1,1,1,1,0],[1,1,0,0,0,1],[1,0,0,1,0,1],[1,0,1,0,0,1],[1,0,0,0,1,1],[0,1,1,1,1,1]];
[ [ Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0, 0*Z(2) ], [ Z(2)^0, Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ],
  [ Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0 ], [ Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0 ],
  [ Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, Z(2)^0 ], [ 0*Z(2), Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0 ] ]
gap> G_D := MakeCodeMat(G,B_6,1,5,1,GF(2));
[ <a GF2 vector of length 12>, <a GF2 vector of length 12>, <a GF2 vector of length 12>, <a GF2 vector of length 12>,
  <a GF2 vector of length 12>, <a GF2 vector of length 12> ]
gap> D := GeneratorMatCode(G_D,GF(2));
a linear [12,6,1..2]3..6 code defined by generator matrix over GF(2)
gap> MinimumWeight(D);
[12,6] linear code over GF(2) - minimum weight evaluation
Known lower-bound: 1
There are 2 generator matrices, ranks : 6 6
The weight of the minimum weight codeword satisfies 0 mod 4 congruence
Enumerating codewords with information weight 1 (w=1)
    Found new minimum weight 2
Number of matrices required for codeword enumeration 2
Completed w= 1, 12 codewords enumerated, lower-bound 4, upper-bound 2
Minimum weight: 2
2
gap> CodeWeightEnumerator(D);
x_1^12+6*x_1^10+15*x_1^8+20*x_1^6+15*x_1^4+6*x_1^2+1
gap> Display(D);
a linear [12,6,2]3..6 code defined by generator matrix over GF(2)
gap> Display(G_D);
```

```
 . 1 . 1 . . 1 . 1 . . .
 . . . 1 . . . . 1 . . .
 . . . 1 . 1 . . 1 . 1 .
 1 . 1 . . . . 1 . 1 . .
 . . 1 . . . . . . 1 . .
 . . 1 . 1 . . . . 1 . 1
gap> G_D2 := MakeCodeMat(G,B_6,1,5,2,GF(2));
[ <a GF2 vector of length 12>, <a GF2 vector of length 12>, <a GF2 vector of length 12>, <a GF2 vector of length 12>,
  <a GF2 vector of length 12>, <a GF2 vector of length 12> ]
gap> Display(G_D2);
 1 . . 1 1 . . 1 . 1 1 .
 1 . 1 1 . 1 1 . . . . 1
 . 1 1 . . 1 . 1 1 . 1 .
 . 1 . 1 1 . 1 . . 1 1 .
 1 . . . . 1 1 . 1 1 . 1
 . 1 1 . 1 . . 1 1 . . 1
gap> D2 := GeneratorMatCode(G_D2,GF(2));
a linear [12,6,1..4]3..4 code defined by generator matrix over GF(2)
gap> MinimumWeight(D2);
[12,6] linear code over GF(2) - minimum weight evaluation
Known lower-bound: 1
There are 2 generator matrices, ranks : 6 6
The weight of the minimum weight codeword satisfies 0 mod 4 congruence
Enumerating codewords with information weight 1 (w=1)
    Found new minimum weight 6
    Found new minimum weight 4
Number of matrices required for codeword enumeration 2
Completed w= 1, 12 codewords enumerated, lower-bound 4, upper-bound 4
Minimum weight: 4
4
gap> CodeWeightEnumerator(D2);
x_1^12+15*x_1^8+32*x_1^6+15*x_1^4+1
gap> IsEquivalent(D2,C4);
true
gap>
```

Appendix II

Example of Gaborit-Otmani $(f_1, a, 3)$ Experimental Construction for $n = 32$ with orthogonal matrices B, $a = 3,\ 11$, and 19.

```
lab47:~~$ gap

            #########           ######         ###########           ###
          ##############        ######         ############        #####
         ###############        ########       ##############      ######
        ################        ########       #####   ######      ######
       ######          #        #########      #####    #####     #######
      ######                   ##########      #####    #####    ########
      #####                    ##### ####      #####   ######   #########
      ####                    #####  #####     ##############   ###  ####
      #####     ########      ####    ####     ###########    ####  ####
      #####     ########     #####    #####    ######        ####   ####
      #####     ########     #####    #####    #####       ##############
       #####      #####     ################    #####       ##############
       ######     #####     ################    #####       ##############
       ################    ##################   #####               ####
        ###############    #####        #####   #####               ####
          #############    #####        #####   #####               ####
           #########      #####          #####  #####               ####

     Information at:  http://www.gap-system.org
     Try '?help' for help. See also  '?copyright' and  '?authors'

   Loading the library. Please be patient, this may take a while.
GAP4, Version: 4.4.10 of 02-Oct-2007, i486-pc-linux-gnu-i486-linux-gnu-gcc
Components:  small 2.1, small2 2.0, small3 2.0, small4 1.0, small5 1.0, small6 1.0, small7 1.0, small8 1.0, small9
1.0,
             small10 0.2, id2 3.0, id3 2.1, id4 1.0, id5 1.0, id6 1.0, id9 1.0, id10 0.1, trans 1.0, prim 2.1  loaded.
Packages:    CTblLib 1.1.3, TomLib 1.1.2  loaded.
gap> LoadPackage("guava");

   ____                            |
  /           \           /   --+--  Version 3.6
 /      |     | |\\        //|     |
|     _  |     | | \\      // |      GUAVA Group
|      \ |     | |--\\    //--|
 \      ||     | |   \\  //    |
  \___/  \___/ |    \\//     |

true
gap>
gap> # This program executes the algorithm designed
gap> # by Philippe Gaborit and Ayoub Otmani to construct self-dual codes.
gap> # The algorithm also computes the greatest minimum distance
```

```
gap> # between codes to find the strongest self-dual codes.
gap>
gap> # This function computes the dot product of two vectors, vecA and vecB.
gap> DotProd := function(vecA, vecB)
>       local ii, tot;
>
>       tot := vecA[1] * vecB[1];
>       for ii in [2..Length(vecA)] do
>               tot := tot + vecA[ii] * vecB[ii];
>       od;
>
>       return tot;
> end;
function( vecA, vecB ) ... end
gap>
gap> # This function constructs the nxn matrix, M, given a matrix B such
gap> # that B*B^T = c * I, the length of the code, n, and the field in
gap> # The matrix M is constructed by placing copies of B along
gap> # the diagonal until B no longer fits at which point the diagonal
gap> # entries are comprised of a scalar matrix b*I, where b^2 = c.
gap>
gap> Build_M_Mat := function(B, n, field)
>       local b, beta, lambda, basemat,
>               blocks, row, ii, M;
>
>       b := Length(B);
>
>       # Special case (just a scalar mat)...
>       # This creates the matrix M in the case where b >n.
>       if Int(n/b) = 0 then
>               lambda := DotProd(B[1], B[1]);
>               beta := First(Elements(field), x -> x*x = lambda);
>               M := beta * IdentityMat(n, field);
>               return M;
>       fi;
>
>       # Normal Case...
>       # This creates the matrix M in the case where b =< n.
>       basemat := IdentityMat(Int(n/b), field);
>       M := KroneckerProduct(basemat,B);
>
>       if n mod b > 0 then
>               lambda := DotProd(B[1], B[1]);
>               beta := First(Elements(field), x -> x*x = lambda);
```

```
>
>               for row in M do
>                    Append(row, List([1..(n mod b)], x -> Zero(field)));
>               od;
>
>               for ii in [1..(n mod b)] do
>                     row := List([1..n], x -> Zero(field));
>                     row[Int(n/b)*b+ii] := beta;
>                     Add(M, row);
>               od;
>       fi;
>
>       return M;
> end;
function( B, n, field ) ... end
gap>
gap> # Build up the permutation matrix Pi, representing
gap> # how the function pi permutes the elements of Z_n
gap> Build_Pi_Mat := function(pi, n, field)
>       local perm, images;
>
>       images := List([0..(n-1)], x -> pi(x));
>       images := List(images, x -> x + 1);
>
>       perm := PermList(images);
>       return PermutationMat(perm, n, field);
> end;
function( pi, n, field ) ... end
gap>
gap> # This function executes the algorithm for creating
gap> # a new self-dual generating matrix for another self-dual code.
gap> # Given an initial self-dual generating matrix G, the previously
gap> # mentioned matrix B, three parameters, f, a, and r, and the field.
gap> MakeCodeMat := function(G, B, f, a, r, field)
>       local M, n, pi, Pi, PiFuncs, Mats;
>
>       if not f in [1..3] then
>               Print("Invalid input for f!\n");
>               return fail;
>       fi;
>
>       n := Length(G[1]);
>       M := Build_M_Mat(B, n, field);
>
```

```
>       if not GCD_INT(a,n) = 1 then
>               Print("Error: n and a must be coprime!");
>               return fail;
>       fi;
>
>       if f = 1 then
>               pi := x -> a*(x+1) mod n;
>               Pi := Build_Pi_Mat(pi, n, field);
>
>               return  G * ((M * Pi) ^ r);
>       fi;
>
>       if f = 2 then
>               PiFuncs := List([1..r], i -> (x -> (a^i)*(x+1) mod n));
>       fi;
>
>       if f = 3 then
>               PiFuncs := List([1..r], i -> (x -> (a^i)*(x+i) mod n));
>       fi;
>
>       Mats := List(PiFuncs, p -> M * Build_Pi_Mat(p, n, field));
>       return G * Product(Mats);
> end;
function( G, B, f, a, r, field ) ... end
gap>
gap> G := Build_M_Mat([Z(2)*[1,1]],16,GF(2));
[ <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>,
  <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>,
  <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>,
  <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32> ]
gap> B:= Z(2) * [[0,1,1,1],[1,0,1,1],[1,1,0,1],[1,1,1,0]];
[ [ 0*Z(2), Z(2)^0, Z(2)^0, Z(2)^0 ], [ Z(2)^0, 0*Z(2), Z(2)^0, Z(2)^0 ], [ Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0 ],
  [ Z(2)^0, Z(2)^0, Z(2)^0, 0*Z(2) ] ]
gap> G_C := MakeCodeMat(G,B,1,3,3,GF(2));
[ <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>,
  <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>,
  <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>,
  <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32> ]
gap> C := GeneratorMatCode(G_C,GF(2));
a linear [32,16,1..8]5..10 code defined by generator matrix over GF(2)
gap> Display(G_C);
```

```
. 1 . 1 1 . 1 1 . 1 1 . . 1 1 1 . 1 1 1 1 . 1 . 1 . . 1 1 . . .
 . . . 1 1 . . 1 . 1 . 1 1 1 1 . 1 1 1 . . 1 1 . 1 1 . 1 1 . 1 .
 1 . 1 . 1 . . 1 1 . . . . 1 . 1 1 . 1 1 . 1 1 . . 1 1 1 . 1 1 1
 . 1 1 . 1 1 . 1 1 . 1 . . . . 1 1 . . 1 . 1 . 1 1 1 1 . 1 1 1 .
 . 1 1 . . 1 1 1 . 1 1 1 1 . 1 . 1 . . 1 1 . . . . 1 . 1 1 . 1 1
 . 1 . 1 1 1 1 . 1 1 1 . . 1 1 . 1 1 . 1 1 . 1 . . . . 1 1 . . 1
 1 . . . . 1 . 1 1 . 1 1 . 1 1 . . 1 1 1 . 1 1 1 1 . 1 . 1 . . 1
 1 . 1 . . . . 1 1 . . 1 . 1 . 1 1 1 1 . 1 1 1 . . 1 1 . 1 1 . 1
 . 1 1 1 1 . 1 . 1 . . 1 1 . . . . 1 . 1 1 . 1 1 . 1 1 . . 1 1 1
 1 1 1 . . 1 1 . 1 1 . 1 1 . 1 . . . . 1 1 . . 1 . 1 . 1 1 1 1 .
 1 . 1 1 . 1 1 . . 1 1 1 . 1 1 1 1 . 1 . 1 . . 1 1 . . . . 1 . 1
 1 . . 1 . 1 . 1 1 1 1 . 1 1 1 . . 1 1 . 1 1 . 1 1 . 1 . . . . 1
 1 . . 1 1 . . . . 1 . 1 1 . 1 1 . 1 1 . . 1 1 1 . 1 1 1 1 . 1 .
 1 1 . 1 1 . 1 . . . . 1 1 . . 1 . 1 . 1 1 1 1 . 1 1 1 . . 1 1 .
 . 1 1 1 . 1 1 1 1 . 1 . 1 . . 1 1 . . . . 1 . 1 1 . 1 1 . 1 1 .
 1 1 1 . 1 1 1 . . 1 1 . 1 1 . 1 1 . 1 . . . . 1 1 . . 1 . 1 . 1
gap> MinimumWeight(C);
[32,16] linear code over GF(2) - minimum weight evaluation
Known lower-bound: 1
There are 2 generator matrices, ranks : 16 16
The weight of the minimum weight codeword satisfies 0 mod 4 congruence
Enumerating codewords with information weight 1 (w=1)
    Found new minimum weight 10
    Found new minimum weight 8
Number of matrices required for codeword enumeration 2
Completed w= 1, 32 codewords enumerated, lower-bound 4, upper-bound 8
Termination expected with information weight 2 at matrix 1

------------------------------------------------------------------------
Enumerating codewords with information weight 2 (w=2) using 1 matrices
Completed w= 2, 120 codewords enumerated, lower-bound 8, upper-bound 8
------------------------------------------------------------------------
Minimum weight: 8
8
gap> CodeWeightEnumerator(C);
x_1^32+364*x_1^24+2048*x_1^22+6720*x_1^20+14336*x_1^18+18598*x_1^16+14336*x_1^14+6720*x_1^12+2048*x_1^10+364*x_1^8+1
gap> G_C2 := MakeCodeMat(G,B,1,11,3,GF(2));
[ <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>,
  <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>,
  <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>,
  <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32> ]
gap> Display(G_C2);
```

```
 . 1 1 . 1 . 1 . . 1 . 1 . 1 1 1 1 . . 1 1 . 1 1 . 1 1 1 1 . . .
 . 1 . 1 . 1 1 . . . . 1 1 1 1 . 1 1 . 1 1 . . 1 1 1 1 . 1 . 1 .
 1 . 1 1 . 1 1 1 1 . . . . 1 1 . 1 . 1 . . 1 . 1 . 1 1 1 1 . . 1
 1 . . 1 1 1 1 . 1 . 1 . . 1 . 1 . 1 1 . . . . 1 1 1 1 . 1 1 . 1
 . 1 . 1 . 1 1 1 1 . . 1 1 . 1 1 . 1 1 1 1 . . . . 1 1 . 1 . 1 .
 . . . 1 1 1 1 . 1 1 . 1 1 . . 1 1 1 1 . 1 . 1 . . 1 . 1 . 1 1 .
 1 . . . . 1 1 . 1 . 1 . . 1 . 1 . 1 1 1 1 . . 1 1 . 1 1 . 1 1 1
 1 . 1 . . 1 . 1 . 1 1 . . . . 1 1 1 1 . 1 1 . 1 1 . . 1 1 1 1 .
 1 . . 1 1 . 1 1 . 1 1 1 1 . . . . 1 1 . 1 . 1 . . 1 . 1 . 1 1 1
 1 1 . 1 1 . . 1 1 1 1 . 1 . 1 . . 1 . 1 . 1 1 . . . . 1 1 1 1 .
 1 . 1 . . 1 . 1 . 1 1 1 1 . . 1 1 . 1 1 . 1 1 1 1 . . . . 1 1 .
 . 1 1 . . . . 1 1 1 1 . 1 1 . 1 1 . . 1 1 1 1 . 1 . 1 . . 1 . 1
 . 1 1 1 1 . . . . 1 1 . 1 . 1 . . 1 . 1 . 1 1 1 1 . . 1 1 . 1 1
 1 1 1 . 1 . 1 . . 1 . 1 . 1 1 . . . . 1 1 1 1 . 1 1 . 1 1 . . 1
 . 1 1 1 1 . . 1 1 . 1 1 . 1 1 1 1 . . . . 1 1 . 1 . 1 . . 1 . 1
 1 1 1 . 1 1 . 1 1 . . 1 1 1 1 . 1 . 1 . . 1 . 1 . 1 1 . . . . 1
gap> C2 := GeneratorMatCode(G_C2,GF(2));
a linear [32,16,1..8]5..10 code defined by generator matrix over GF(2)
gap> MinimumWeight(C2);
[32,16] linear code over GF(2) - minimum weight evaluation
Known lower-bound: 1
There are 2 generator matrices, ranks : 16 16
The weight of the minimum weight codeword satisfies 0 mod 4 congruence
Enumerating codewords with information weight 1 (w=1)
    Found new minimum weight 10
    Found new minimum weight 8
Number of matrices required for codeword enumeration 2
Completed w= 1, 32 codewords enumerated, lower-bound 4, upper-bound 8
Termination expected with information weight 2 at matrix 1
------------------------------------------------------------------------
Enumerating codewords with information weight 2 (w=2) using 1 matrices
Completed w = 2, 120 codewords enumerated, lower-bound 8, upper-bound 8
------------------------------------------------------------------------
Minimum weight: 8
8
gap> CodeWeightEnumerator(C2);
x_1^32+364*x_1^24+2048*x_1^22+6720*x_1^20+14336*x_1^18+18598*x_1^16+14336*x_1^14+6720*x_1^12+2048*x_1^10+364*x_1^8+1
gap> IsEquivalent(C,C2);
true
gap> G_C3 := MakeCodeMat(G,B,1,19,3,GF(2));
[ <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>,
  <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>,
  <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>,
  <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32>, <a GF2 vector of length 32> ]
```

```
gap> C3 := GeneratorMatCode(G_C3,GF(2));
a linear [32,16,1..8]5..10 code defined by generator matrix over GF(2)
gap> MinimumWeight(C3);
[32,16] linear code over GF(2) - minimum weight evaluation
Known lower-bound: 1
There are 2 generator matrices, ranks : 16 16
The weight of the minimum weight codeword satisfies 0 mod 4 congruence
Enumerating codewords with information weight 1 (w=1)
    Found new minimum weight 8
Number of matrices required for codeword enumeration 2
Completed w= 1, 32 codewords enumerated, lower-bound 4, upper-bound 8
Termination expected with information weight 2 at matrix 1
--------------------------------------------------------------------------
Enumerating codewords with information weight 2 (w=2) using 1 matrices
Completed w= 2, 120 codewords enumerated, lower-bound 8, upper-bound 8
--------------------------------------------------------------------------
Minimum weight: 8
8
gap> CodeWeightEnumerator(C3);
x_1^32+364*x_1^24+2048*x_1^22+6720*x_1^20+14336*x_1^18+18598*x_1^16+14336*x_1^14+6720*x_1^12+2048*x_1^10+364*x_1^8+1
gap> IsEquivalent(C3,C2);
true
gap>
```

BIBLIOGRAPHY

[1] E. R. Berlekamp, F. J. MacWilliams and N. J. A. Sloane, Gleason's theorem on self-dual codes, *IEEE Trans. Inform. Theory*, vol. IT-18, No. 3, 1972.

[2] S. Buyuklieva, On the Binary Self-Dual Codes with an Automorphism of Order 2, *Designs, Codes and Cryptography*, vol. 12, 39-48, Kluwer Academic Publishers, Boston, 1997.

[3] J.C. Carlach, A. Otmani, C. Vervoux, A new scheme for building good self-dual block codes, *Proceeding of 2000 IEEE International Symposium on Information Theory (ISIT 2000)*, Sorento, Italy, 25-30 June 2000, p. 476.

[4] J. H. Conway and N. J. A. Sloane, A new upper bound on minimal distance of self-dual codes, *IEEE Trans. Inform. Theory*, vol. 36, 1990, pp. 1319-1333.

[5] A new family of block Turbo-Codes, *Proceeding of the 13th Applicable Algebra in Engineering Communication and Computing (AAECC 13)*, Hawaii, USA, 14-19 November 1999, p. 15.

[6] P. Gaborit and A. Otmani, Experimental construction of self-dual codes, *Finite Fields and Their Applications 9*, 2003, 372-394.

[7] M. Harada, M. Kitazume and A. Munemasa, Note On A 5-design Related to An Extremal Doubly Even Self-Dual Code of Length 72, *Journal of Combinatorial Theory*, Series A 107, 143-146, 2004.

[8] W. C. Huffman, On the classification and enumeration of self-dual codes, *Finite Fields and Their Applications 11*, 2005, 451-490.

[9] W.P. Johnson, The curious history of Faá di Brunos's formula, *The mathematical Association of America Monthly 109*, March 2002, 217-234.

[10] G. T. Kennedy, V. Pless, On Designs And Formally Sefl-Dual Codes, *Designs, Codes and Cryptography*, vol. 4, 43-55, Kluwer Academic Publishers, Boston, 1994.

[11] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland, Amsterdam, 1977.

[12] F. J. MacWilliams, C. L. Mallows, and N. J. A. Sloane, Generalizations of Gleason's Theorem on Weight Enumerators of Self-Dual Codes, *IEEE Trans. Inform. Theory*, vol. IT-18, No. 6, 1972.

[13] C. L. Mallows and N. J. A. Sloane, An Upper bound for self-dual codes, *Information and Control*, vol. 22, No. 2, 1973.

[14] V.S. Pless and W.C. Huffman editors, *The Handbook of coding theory*, vol I and II, Elsevier Sc. Publ., Amsterdam, 1998.

[15] E. M. Rains, Shadow bounds for self-dual codes, *IEEE Trans. Inform. Theory*, vol. 44, No. 1, 1988.

[16] C.E. Shannon, "A Mathematical Theory of Communication", *Bell System Technical Journal*, vol. 27, I948, pp. 379-423, 623-656.

[17] N.J.A. Sloane, Gleason's Theorem on Self-Dual Codes and Its Generalizations, *arXiv:math/0612535v1*

[18] M. Armentrout, M. Pitluck, R. Rohatgi, J. Thomas, S. Kalaycioglu, K. Lux, Three approaches to self-dual code construction, *2008 VIGRE Arizona Summer Program Report*, Arizona, 2008.

[19] H.N. Ward, A restriction on the weight enumerator of a self-dual code, *J. Combin. Theory*, Ser. A 21, 1976, 253-255.

[20] E.T. Whittaker and G.N. Watson, *A Course in Modern Analysis*,4 Ed., Cambridge University Press, Cambridge, 1990.

VITA

Rini Oktavia was born in Banda Aceh, Indonesia in 1970, the daughter of Farida Boerhan and Ramli Ibrahim. After completing her work at SMAN 3 Banda Aceh, Indonesia, in 1988, she entered the Bandung Institute of Technology in West Java, Indonesia. She received the degree of Sarjana Sains from the Bandung Institute of Technology in October, 1993. In August, 1995, she attended the graduate program in Mathematics at the Bandung Institute of Technology and graduated with Master Sains degree in October, 1998. During her graduate study, in December, 1995, she was accepted as a lecturer at the Education University of Indonesia in Bandung and worked there until June, 2003. In July, 2003 she moved to Syiah Kuala University in Banda Aceh, Indonesia. In August, 2007, she got a fellowship from Ford Foundation International Fellowship Program to attend the graduate program in Mathematics at the University of Texas at Austin.

Permanent Address: Jl. Tgk. Di Blang No. 66
Banda Aceh, 23123
NAD, Indonesia.

This report was typed by the author.